A Real-Time Robotic Platform for Pipeline Inspection

by

Ryan J. Fish

B.S., Massachusetts Institute of Technology (2015)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Author .....
Department of Mechanical Engineering
May 12, 2017

Certified by .....
Kamal Youcef-Toumi
Professor
Thesis Supervisor

Accepted by .....
Rohan Abeyaratne
Chairman, Department Committee on Graduate Students

A Real-Time Robotic Platform for Pipeline Inspection

by

Ryan J. Fish

Submitted to the Department of Mechanical Engineering
on May 12, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

Pipelines are used around the world to transport raw materials, waste products, and, critically, potable water. Protecting the pipes from the elements often leaves them difficult to inspect for leaks and corrosion, which can cause costly, even deadly, damage. Currently, typical inspection methods are costly, interrupt service, and have highly limited inspection range. This thesis details the continuing development of a robotic platform capable of actively maneuvering inside an in-service, potable-water pipe, for the purpose of providing continuous, autonomous, long-range inspection of a pipe network. Complete inspection of municipal water pipelines requires a compact robot capable of maneuvering junctions around 100mm in diameter, with flows that can exceed 1m/s. This work focuses on several additions to prior work at the Massachusetts Institute of Technology, which developed a hull and planar propulsion system. The addition of ailerons allows full 3-dimensional control of the robot. A custom-built, wireless embedded controller runs a customized real-time OS to provide closed-loop control, as well as datalogging and remote access. A generic task architecture is designed to simplify the addition of real-time software modules.

Thesis Supervisor: Kamal Youcef-Toumi
Title: Professor

Acknowledgments

I first must give thanks to my incredible advisor, Prof. Kamal Youcef-Toumi. His breadth of knowledge is always an inspiration, his kindness and generosity lighten the days of everyone who meets him. It has been my privilege to have been his student these past two years.

Thanks to the experience and ever-present helpfulness of my colleague You Wu, I was able to continue his work on this robot. His guidance saved me many hours of making mistakes he had already learned from. His creative thinking sparked new ideas and directions, and his recommendations were always serious and thoughtful.

I also want to thank my friends and family, who build homes for me wherever we go. Shared struggle provides the greatest comfort, and shared triumph the greatest joy. Stay strong, travel far, live long and prosper.

Finally, my deepest thanks goes to my closest friend and companion, Hunter Guarino. Without her presence, no matter the distance in miles, my life would be a thin image of where I am today. Her patience and understanding are unparalleled, her curiosity is inspiring. Here's to many more adventures!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter outlines the current functions of pipelines in modern society, the problems caused by leaks, and the issues faced with contemporary inspection solutions. Chapter 2 outlines the physical requirements for an inspection robot and this work's addition of out-of-plane actuation to this lab's previous work. Chapter 3 sets out the electrical requirements and analyzes the design of an embedded controller circuit board to run the robot autonomously. Chapter 4 discusses the software design that was implemented for control and autonomy. Finally, Chapter 5 provides conclusions and lessons learned, and lays out recommendations for the future progress of this work.

1.1 Pipelines: A Modern Necessity

Pipelines provide safe, low-maintenance, always-on transportation for the fluids that power modern life. The Oil and Gas Journal estimates 14.1 billion barrels (bbl) of oil were pumped through pipelines across the US alone in 2012[1]. For goods requiring high transport volumes, such as water and oil, dedicated infrastructure is the only economical way to provide continuous and on-demand resources. The US Federal Energy Regulatory Commission estimates that 17% of the nation's domestically transported goods move via pipelines while claiming only 2% of the overall freight costs[2].

Pipelines also provide safety improvements versus over-land transport. Tanker trucks increase traffic congestion and move toxic, volatile, flammable goods in close proximity to people and infrastructure. Rail-based transport of oil products has increased in recent years, to the growing concern of regulators. Convoys of over 100 cars carry oil products via rail that is dangerously close to populated areas. This danger was highlighted in Lac Megantic, Canada when a parked train started rolling downhill, without its operator, before crashing into the town below, killing 42 people and causing a major fire[1]. Pipelines are generally designed for transporting a single, known good, and integrate safety measures commensurate with the hazards.

As for drinking water, municipalities with high population density get several benefits from a centralized pipeline distribution. Drilling wells is costly and unscalable, much more expensive than a few meters of pipe from a street to a home. A well with the capacity to service an entire office building would need to be massive, further increasing drilling costs. The size of a city and its rate of water demand also contribute to aquifer draw-down, increasing pumping costs and minimum well-depth[3]. A centralized source allows for uniform water-quality across a city, where local groundwater quality may vary with proximity to pollution, fresh-, and salt-water sources. An EPA assessment of water system infrastructure identified approximately 52,000 municipal water systems, and 21,400 non-profit private water systems across the US in 2011 [4].

1.2 Motivation for Pipeline Inspection

With pipelines performing vital functions in modern society, including transporting dangerous materials and supporting modern cities with clean, plentiful water, there is a direct need for preventative maintenance and repair. Problems with pipelines typically manifest as leaks or blockages, both of which can cause considerable problems. For oil and other hazardous material pipelines, the problems with leaks are direct, with the release of material presenting a serious environmental and public-health hazard. Partial blockages increase pumping costs, and total blockages can shut down facilities and endanger the pipeline infrastructure.

Figure 1-1: Carrer Ganduxer, Barcelona, after water mains burst[7]

For drinking water, leaks pose a less direct issue, though no less serious. The American Society of Civil Engineers (ASCE) estimates 6 billion gallons of treated drinking water is lost daily, 15% of daily usage, in the US across 1 million miles of pipes[5]. Small leaks can erode and soften the ground around a pipe, leading to increased loading from the ground, buildings and/or street above it. About 240,000 water mains fail annually in the US. The failure of a large water main can be extremely expensive; a 3-meter diameter steel water supply line failed in 2010 in Boston, MA, causing 10,000,000 m^3 of water to be lost, resulting in $85 million in damage. The repair took 60 hours [6]. Much smaller leaks happen more frequently, and can still destroy entire streets such as Carrer Ganduxer in Barcelona in 2016, shown in Figure 1-1. Leaks also present openings for contaminants to enter the water supply, risking public health.

The complexity of a city water system makes detecting leaks difficult and determining their location even more so. The water utility may know which streets or houses are affected by an issue but the problem could lie anywhere along the branch servicing those locations, stretching an entire block or more. Worse still, small leaks may not cause detectable loss individually but can make up a significant portion of

total system water loss due to their frequency and the duration they are active before repair.

1.3 Current Solutions, Current Problems

Current solutions to pipeline inspection vary by transport material, pipeline usage and resources. Operators of pipelines carrying hazardous materials are required by government regulations to clean and inspect both the inside and outside of their pipelines[8]. Industries using these pipelines have built inspection and cleaning into their operations, designing pipelines to enable rapid entry of cleaning devices, sometimes known as pipeline inspection gauges or "pigs," into the line. A pipeline crossover is shown in Figure 1-2, where the flow can be momentarily turned off, a pig inserted, and flow resumed until the pig can be retrieved at the next station. An inspection pig may simply be a foam slug the same diameter as the pipe that is inserted into the line, which is then pressurized, driving the slug downstream. The foam collapses as it contacts buildup, and when it is retrieved at the end of the inspection the pipeline operator can determine whether a cleaning run is required. Others have aggressive scraping tools built into them, designed to clear the pipe out to its original ID, like the one shown in Figure 1-3 from the Trans-Alaskan Pipeline.

There are active pigs that may be equipped with inspection cameras, cleaning brushes, on-board propulsion, and even facilities to recoat the inside of the pipe with protective sealants[11]. Active pigs are nearly always tethered to an operations base, and so have limited range. Federal regulations ensure that these operations happen frequently enough to identify pipe wear, clogging, and corrosion before it becomes an issue.

Municipal water supplies do not have the same kind of federal regulation driving inspection. These pipelines are built to minimize cost, visibility and risk of damage, and so are relegated to subsurface utilities. They typically lack convenient means of entry for a pig, though solutions do exist designed for entry through fire hydrants[12]. Pigging also requires a complete shutdown of the line, as dislodged corrosion and

Figure 1-2: A crossover station to allow for temporary flow stoppage and pig insertion[9]

Figure 1-3: A cleaning pig used on the Trans-Alaskan Pipeline[10]

deposits contaminate the water supply. The end of the zone under inspection must be opened and allowed to drain, which also provides for pig retrieval. Otherwise, the capabilities afforded by pigs are comprehensive, including the aforementioned repair capabilities. Other options for inspection and leak detection include acoustic monitoring, either with a manned operator or using mounted sensors, which listen for acoustic vibrations generated by escaping liquid, or using a tracer gas such as H_2, inserted upstream, and a detector to see how much leaks to the surface.

There are a few examples of tetherless commercial robotic inspection tools. One of these, the Smartball™ by Pure Technologies™ is a passive foam ball with an electronic core containing Inertial Measurement Units (IMUs) and acoustic sensing equipment. The soft foam allows it to deform to bypass obstructions and junctions. However, provisions must be made downstream to direct its course, since it passively follows the flow. A special retrieval outlet must also be integrated into the pipeline. Pipetel Technologies' Explorer robotic inspection tool is the most advanced on the market. It combines advanced pipeline health sensing and optical inspection tools with a traction-based maneuvering system. It includes inline-charging to increase its range, and can navigate T-junctions, elbows and vertical segments of piping. It's main weakness is that it is of fixed diameter; it cannot inspect the wide variation of pipe diameters that comprise an urban water supply network.

With the exception of permanently mounted sensors, all solutions require significant human input and expertise, as well as system shutdowns and wasteful draining and flushing. Every contemporary solution can be deployed only along a limited stretch of pipeline, requiring prior knowledge of at-risk zones or potentially wasting money on inspecting pipes that have no issues. Municipalities, with limited funds to spend towards preventative maintenance, accordingly wait to fix obvious problems rather than spend money on ineffectual and disruptive early detection attempts.

1.4 Overview of Research Direction

This research aims to build a robotic inspection platform designed to continuously and autonomously navigate and inspect municipal drinking water supply pipelines. A small, self-propelled vessel provides several capabilities that improve upon the status quo.

The first improvement is to develop the robot such that it can remain inside the pipeline. By placing the inspection and sensing unit inside the pipes, no costly digging, street closure, or repaving has to occur for each inspection. This also avoids the shutdown and draining procedures that are required in current inspection methods.

The second improvement is adding sensing capability to allow location awareness. For an autonomous inspection platform to provide useful information to the pipeline operator, inspection data must have accurate positioning. Unnecessary service interruptions are reduced by insuring only zones that need cleaning and repair are shut down.

A third improvement of this method is that it provides continuous monitoring, rather than problem localization after leaks are already occurring. Overall safety of the system is improved, as early detection and repair of leaks minimizes contamination and property damage caused by flooding and sinkholes. Operators can also more easily monitor changing pipeline conditions through time, enabling not only failure prediction and budgeting but also root cause analysis. This can inform operators of better pipeline designs, as well as suggest improvements for impinging infrastructure which influence pipeline longevity.

A fourth improvement is that an autonomous robot doesn't need a highly trained operator to control its movements. The untethered design and difficult wireless environment preclude solutions that have human-in-the-loop control.

1.5 Overview of Thesis Work

To further these design goals, this work introduces an integrated embedded controller to the robot. With more sophisticated electronics on board, a self-maintaining robotic system can be developed. With the relatively minor installation of a charging and data uplink station, the robot could run indefinitely. Greatly increased processing power is added which allows for more complex software, increasing the robot's autonomy and navigation capabilities. An IMU provides kinematic state estimation, and the processor power leaves room for additional sensing and sensor fusion. This work also develops an aileron control surface to allow for increased workspace and controllability.

These improvements bring the Mechatronics Research Lab (MRL) at the Massachusetts Institute of Technology (MIT) closer to a robotic inspection solution that is capable of operating entirely within a pipeline, with no human interaction except to act upon uploaded data.

Chapter 2

Physical Design

This chapter discusses the requirements for an autonomous inspection robot to physically navigate a pipeline. The previous work on the robot platform is reviewed, and limitations considered. Several candidate actuators are considered to add out-of-plane motion to the robot, and an optimal design is selected.

2.1 Requirements

The robot design starts by considering the constraints on a device with a practical application. Constraints come from several sources, including the typical design of pipelines, the available manufacturing equipment, and physical optimality for navigation.

The American Water Works Association (AWWA) outlines standards for municipal water supply pipelines in the United States, which are frequently followed by operators in this region. Their recommendations are a good starting point to understand the physical environment of the robot.

Municipal water supplies are typically sized to meet either industrial demands in areas with heavy industry, or are sized to meet firefighting requirements in predominantly residential and light commercial areas. Residential requirements specify a minimum of 6in diameter supply pipes to provide guaranteed firefighting flow rates and pressures given adequate supply capability. However, some municipalities with

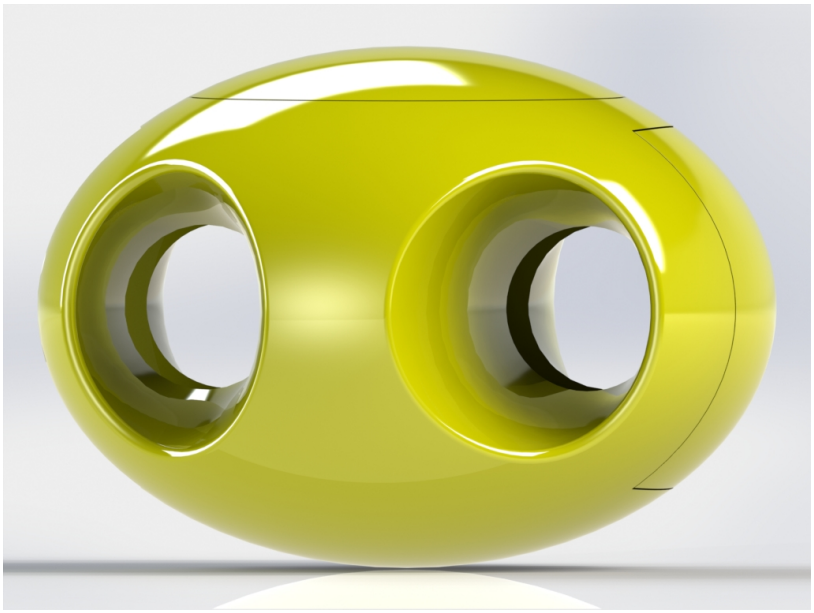

Figure 2-1: Pipeline inspection robot hull as designed by You Wu in [14]

low population densities may have 4in pipes to reduce the volume of stagnant water, which can become a health concern[13].

The previous research designed the robot hull, pictured in Figure 2-1, to fit within a 4in pipe, which balances versatility throughout a practical system, and the need for internal volume to fit actuators and electronics. Empirically, it was determined that 4in nominal internal diameter (ID) fittings and elbows are smaller than 4in, so the hull fits within the recommended 3.35in bounding sphere.

Another constraint comes from drag induced through moving flows of fluid. As the fluid approaches the volume occupied by the robot, it must speed up to conserve volumetric flow rate. While the propulsion method works to relieve this condition, the required size and power of the actuator can be reduced by shrinking the cross-section of the robot[14]. Therefore, the previous work reduced the robot's shape from a sphere to an oblate spheroid with a height of 2.37in, representing a 30% reduction in flow-facing cross-section.

Ideally, the robot should be neutrally buoyant. This property will allow the robot to remain at rest without control effort, and minimizes control effort during motion. The net displacement of the bounding oblate spheroid is 2.27×10^5 mm^3. Subtract-

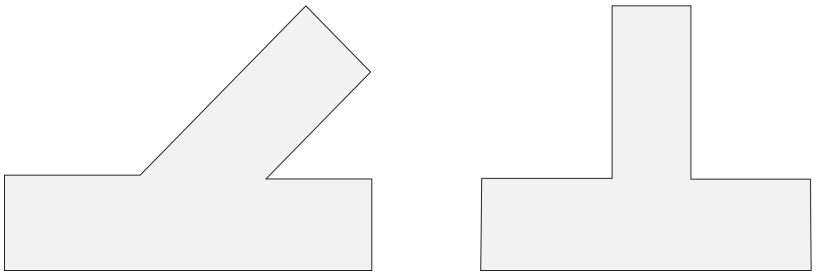

Figure 2-2: Common junctions encountered in a pipeline network include Y-junctions, left, and T-junctions, right

ing the ducts for the motors, which are described in Section 2.2, brings the robot displacement to 1.75×10^5 mm^3, or the equivalent of 175g of water. The new design should maintain the robot at this weight or under, as weight can be added to bring the robot to neutral buoyancy.

The AWWA also recommends that any single endpoint be capable of supplying at least 500gpm to a firefighting unit[13], representing a linear flow rate of just under 1m/s along a 4in diameter pipe. Therefore the robot should be capable of swimming against at least this flow rate. The previous work developed a set of RIM propeller actuators capable of driving the robot up to 0.3m/s[14]. This was considered sufficient for the current stage of testing.

The design of pipe junctions provides dynamic requirements for a free-swimming inspection robot. Typically, joints are designed to be gradual and smooth, so as to provide the lowest "head-loss," or pressure reduction, possible. However, take-offs from a larger supply, especially those added after the installation of the main line, do exist as perpendicularly welded T-junctions without filleting around the edges. This means a robot which can inspect all areas of a pipeline must be able to make 90° turns with a radius of the pipe diameter. More challenging still, some pipelines employ Y-junctions, which are typically a straight section of pipe with a takeoff at a 45° angle. Examples of Y- and T-junctions are shown in Figure 2-2. Ideally, a robot can make the maneuver from one end of the acute angle to the other, where the robot rotates 135° in an arc of the pipe's radius. Furthermore, the last pipe in a distribution network before reaching a home or business will be a dead end because

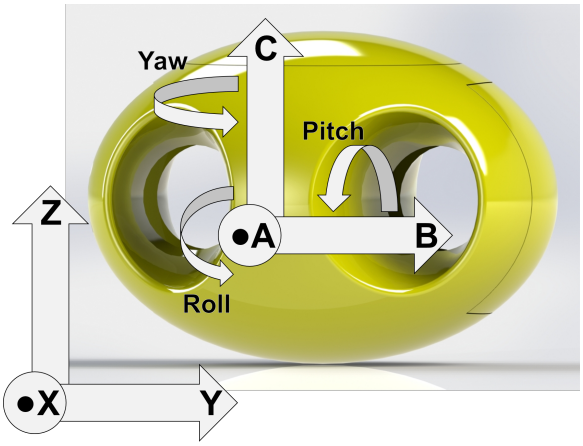

Figure 2-3: Pitch, Yaw, and Roll rotation axes of the robot, considered in the robot's body frame, A,B,C, and a Global X,Y,Z coordinate frame

it will terminate in a meter, or a reduction to a 2in diameter or smaller. A robot should be capable of 180° turns in place, or be able to drive in reverse.

For the robot to change pitch, as defined in Figure 2-3, by 90° while moving at 1m/s in minimum time, such that it can turn through a T-junction without missing it (about 100mm), means it must execute that turn in 100ms. The angular acceleration required to execute this maneuver, assuming the robot has a maximum angular acceleration which is symmetric in the opposite direction, is 1260 rad/s^2.

The robot's environment also provides significant added mass forces, as the robot has to displace water to rotate. In the pitch and roll rotation directions, the robot has to displace the water in the bounding sphere of the robot, as shown in Figure 2-4, which provides a moment of inertia larger than the robot's in these directions, estimated to be 134 kg mm^2. The goal torque to control the pitch of the robot is 0.17 N m, to allow for the desired angular acceleration with the added mass from the displaced water.

2.2 Previous Work

MRL has been developing an autonomous, tetherless inspection robot for several years. The work up to this point has developed several facets of the robot design.

The previous work by You Wu[14] determined optimizations for the shape and size

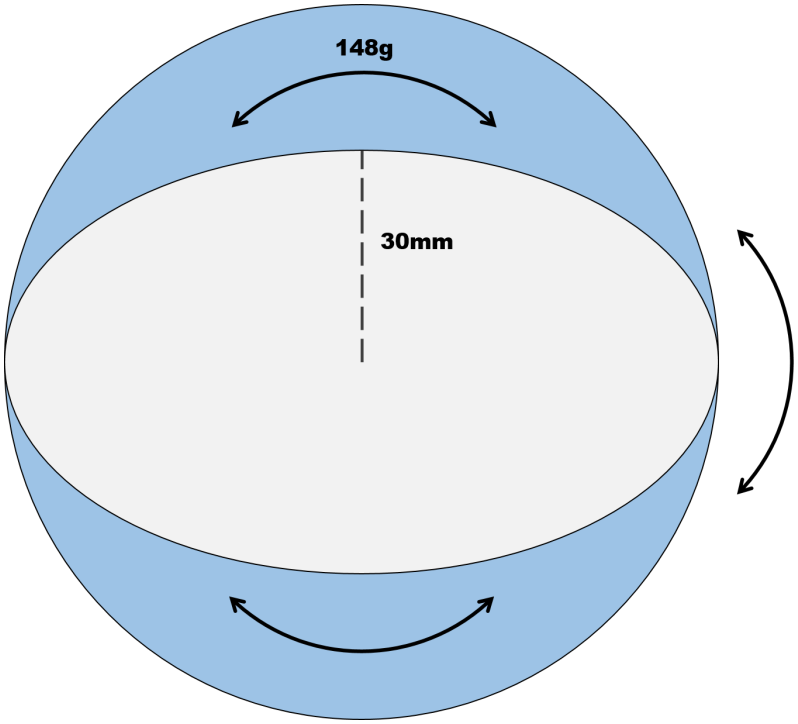

Figure 2-4: The water contained in the bounding sphere of the robot, which must be displaced during pitching or rolling motions, has a mass of 148g. This mass is concentrated over 30mm away from the center of the robot.

of the hull. This was driven particularly by the pipe and pipe connection constraints, and the drag increase with blockage of pipe cross-section.

The same work also developed actuators to provide thrust to move the robot. These actuators took the form of rim propellers, a form of brushless in-runner ducted fan. A section view of the duct through the hull is shown in Figure 2-5. These motors allow for power transmission without physical connection penetrating from inside to outside the hull, meaning an inherently waterproof design. The ducting also provides additional cross-section to allow water to flow by, decreasing drag. Finally, a ducted rim propeller protects the propeller blades from collisions with the pipe walls, ensuring the robot can recover and maintain propulsion even under turbulent flows or after navigational errors[14].

Lacking from the previous work was the ability to control the robots motion out-of-plane, namely, its displacement in Z when starting from an orientation with $C \parallel Z$ (see Figure 2-3 for axis illustration). The thrusters as designed were bidirectional,

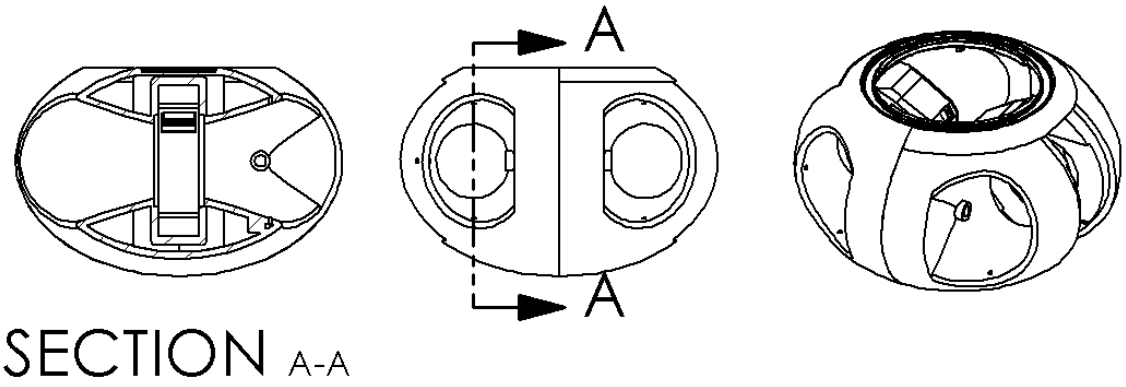

Figure 2-5: Section, side and trimetric view of robot hull. The section view cuts longitudinally through the thruster.

which provided non-holonomic planar control for the robot. Translation along the A axis of the robot was possible, as was yaw, yielding full motion in the XY-plane. No actuation existed that affected the translation along the Z axis, or pitch and roll rotations. This left the robot uncontrollable for perturbations in those directions, and prevented the robot from being able to maneuver a general pipeline network.

2.3 Design of Out-of-Plane Control Surfaces

For a robot in a general pipeline, several factors dictate a need for out-of-plane motion. Pipelines do not necessarily remain at the same elevation, even if the ground surface does. Other utilities, such as drainage, sewage, electric and telecom, may have priority, right of way, or separation requirements that result in the water mains changing elevation. Furthermore, real cities may have surface topography that requires the buried utilities to change in elevation with this topography.

Additional need for out-of-plane motion is the design of junctions in the pipeline. Take-offs from the main supply can happen at any angle relative to the horizontal, as shown in Figure 2-6. A robot needing to navigate an entire network must have a means to provide out-of-plane rotations.

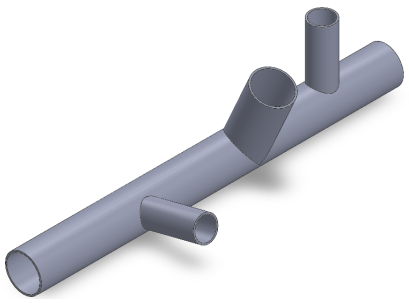

Figure 2-6: Pipeline with multiple take-offs

2.3.1 Considerations

The following describes the options that were evaluated as candidates for out-of-plane motion control of the robot. These options are derived from solutions found in the many industries requiring orientation control, such as the aeronautical and astronautical sectors, and submarine or unmanned underwater vehicle (UUV) sector.

2.3.1.1 Thrust Vectoring Nozzles

Thrust vectoring is a principal that has been applied to great effect in all three reference industries. Its prime motivation is to provide a versatile thrust vector using only one powerful actuator, which can change direction based on inputs from one to two smaller actuators.

Well-known contemporary examples of thrust vectoring in the astronautical industry are the nozzles of rockets. Here, a single powerful engine is used to propel the rocket towards its target but the rocket can experience disturbances such as wind. The nozzle controlling the escape of exhaust gases has relatively small actuators that can direct the direction of these gases. Doing so changes the thrust applied to the base of the rocket, generating a torque about its center of mass. This allows a single, nominally axial actuator to correct for radial disturbances[15].

Other examples of thrust vectoring include the famous Harrier Jump Jet, the first successful, production jet aircraft with Vertical Take-Off and Landing (VTOL) capabilities[16]. Again, the key was keeping the main body of the physically massive actuator static, and adjusting only the direction of the exhaust gases as they exited

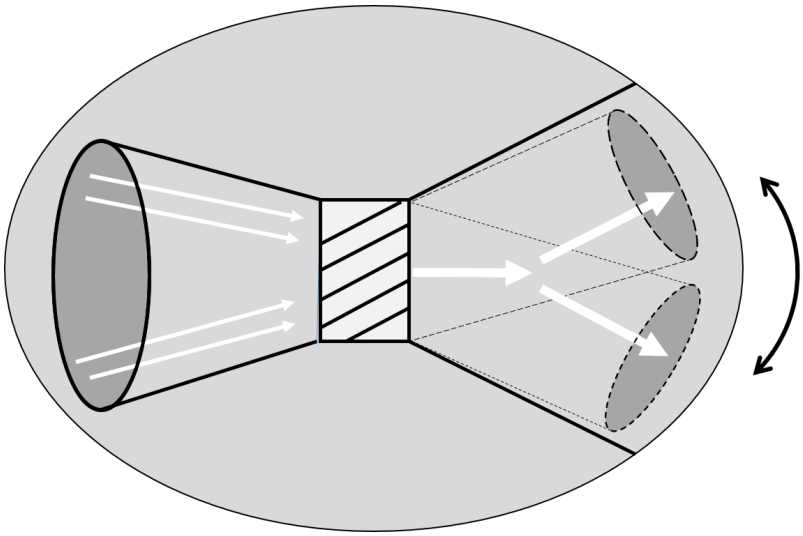

Figure 2-7: Concept drawing of using a thrust vectoring nozzle to provide out-of-plane motion. View shown similar to section view of Figure 2-5

the engine. This allows very small actuators to have fine directional control over massive amounts of thrust, such that the Harrier was able to take off vertically, and quickly transition to horizontal flight, all with a single large actuator.

Applying this concept to the inspection robot, the existing rim propellers take the role of a large, powerful actuator. By influencing the shape of the ducts that guide the water flow, the robot could vector the generated thrust, as shown in Figure 2-7. As the existing setup can already provide yaw control, the nozzles would only need to be pivoted about the B axis. Moving both nozzles in the same direction provides pitch control, and moving them differentially provides roll control. This is sufficient for providing full, non-holonomic controllability of the robot.

To create this thrust vectoring setup, a composite soft/rigid material vectoring nozzle was considered. To maintain watertightness, a soft, deformable rubber compound could be cast between the hard, "static" hull of the main body and the hard, pivoting vector nozzle. An actuator between the hull and the nozzle would then provide relative rotation between the two, changing the thrust vector with respect to the hull. The soft material seal would extend or deform to continue to provide a working seal even with this relative motion.

Challenges with this method include asymmetry, complexity, and space require-

ments. Typically, nozzles in the aeronautics and astronautics sectors provide a final stage of compression in the exhaust flow, increasing its exit velocity and increasing thrust. The nozzle geometry itself has a strong effect on the direction of the exhaust flow[15], and so small changes to the nozzle provide large changes in the thrust vector.

For thrust vectoring to work in this robot, the vectoring nozzle would either need a large range of motion, as seen with the Harrier, or to provide some additional nozzle effect that would entrain the flow along the vector direction even with small displacements. A large range of motion design becomes a concern when considering the sealing soft material would experience 200% elongation or more. The nozzle redesign for increased flow entrainment may have the negative effect of making the propeller thrust asymmetric for respect to forward and reverse thrust. It is desirable to keep the robot as symmetric as possible to maintain the thrusters' ability to provide reverse thrust for yaw control. If the propellers develop direction-dependent maximum thrust, turning at the speeds required to navigate junctions along fast flow will be impeded.

Complexity also became a concern with this design. Sealing the nozzles requires an adequate seal along 2 separate interfaces on each side of the robot, shown in Figure 2-8, totaling approximately 30cm of length. That represents a large seal region, which increases the likelihood of seal failure. Additionally, the mechanism needed to servo the nozzle would be complex, as some sort of drivetrain would be needed to change the axis of nozzle rotation to an axis inside the robot with room for an actuator.

Finally, the deformation required by the nozzle, and the volume swept out during servoing, greatly reduce the usable volume inside the robot. The highly volume-constrained design would not be served well by these side effects.

2.3.1.2 Additional Propeller Axis

Another candidate for out-of-plane motion is an additional ducted rim propeller. Several prototype aircraft were developed in the 1960s using this method for VTOL capabilities. NASA was involved in the development of the XV-5 series aircraft, which utilized a jet for forward thrust, and ducted propellers built into the wings for lift

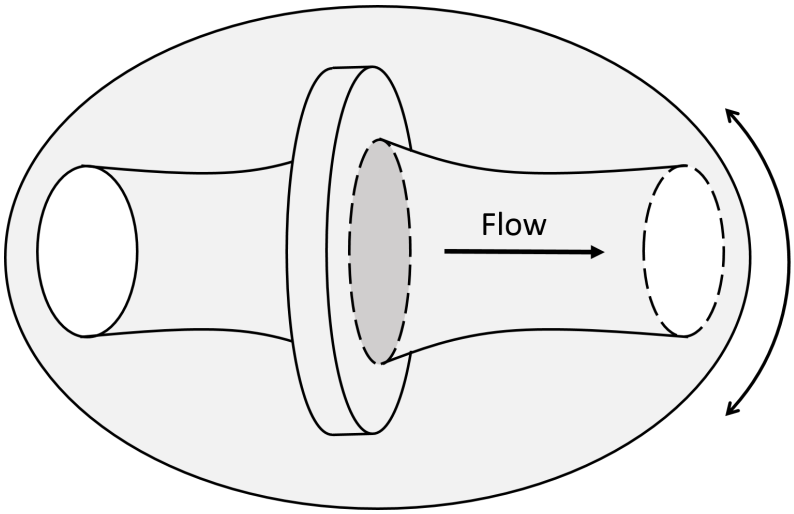

Figure 2-8: The dashed lines show the regions of the vectoring nozzle that would need to be sealed by compliant material

during VTOL.

A report from a test pilot of the XV-5 describes the cross-coupling of dynamics between forward motion and vertical control as "difficult," with "landing precision markedly degraded"[17]. The combination of a cross-flow over a thrust element causes an apparent drag force that creates a pitching moment about the thrust element, called "ram drag"[18]. This results in undesirable cross-coupling.

The same pilot and additional sources agree that at high speeds, the vertical control all but disappears as the slipstream carries air across and away from the ducting surface. This flow impairs the ability of the propeller to generate lift[17][18].

$$Re = \frac{uL}{\nu} \tag{2.1}$$

Water generally has an order of magnitude lower kinematic viscosity number (ν) than air at Standard Temperature and Pressure (STP) as well as at cruise altitude of a typical aircraft. Air has a kinematic viscosity of about 15e-6 m^2/s [19], water has one of about 1e-6 m^2/s [20], both at STP. Air at 12km is 40e-6 m^2/s [21], since pressure falls (air becomes less viscous) faster than temperature (more viscous) with increasing altitude. Therefore, the environment of the robot is an order and a half more viscous than the environment of a test aircraft.

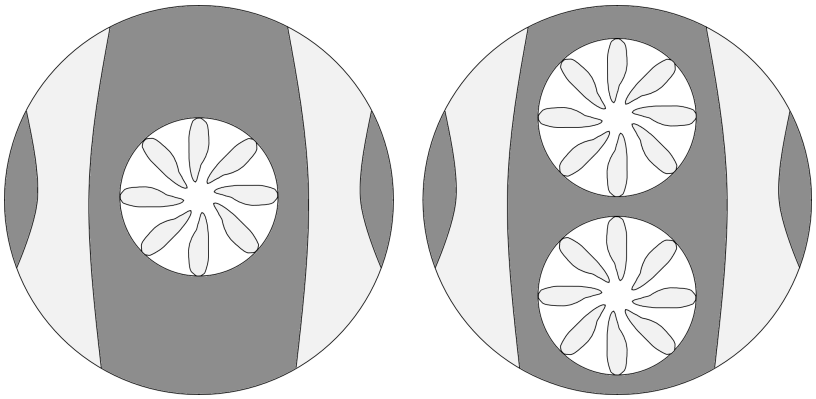

Figure 2-9: Concept drawing of adding a single thruster (left), which is unable to control pitch or roll. Adding two thrusters (right) can control pitch, but roll is still uncontrolled. Top view, cross-section.

The robot would be moving one to two orders of magnitude slower than this test aircraft. The duct diameter would be about two orders of magnitude smaller (the test aircraft described had a duct of 1m[22] and the robot's ducts are about 2cm). The net order of the robot Reynolds number is two to three orders of magnitude lower than the test aircraft, so the expected behavior of a ducted propeller in water with a cross flow is to have a smaller degree of turbulance in the duct. This may indicate the additional duct would function well at all speeds but testing would be needed.

In addition to the above concerns, adding a third ducted propeller would consume even more of the internal volume of the robot. The two main drive ducts occupy more than 5.77×10^4 mm^3 and the motor mounting volume occupies approximately 1×10^4 mm^3 additional volume. This represents over 30% of the internal volume consumed by 2 ducts, though even more of the internal volume is rendered unusable via small gaps around the edges of the motors. An additional motor would decrease the usable internal volume to less than 50% of the bounding volume.

Finally, a single additional propeller as shown in Figure 2-9 would be unable to control the roll or pitch of the robot. While out-of-plane translation is desirable, the robot needs controllability of the rotations and can make due with a non-holonomic method of translation. Pitch could be controlled with a fourth propeller working differentially with the third but the sacrificed volume is not worth the control.

2.3.1.3 Control Moment or Reaction Gyroscope

Spacecraft have used gyroscopes as means of attitude control for decades. As simple, robust ways to generate torques in free-bodied craft, they are favored for not requiring consumable fuels. The first space station, Skylab, included control-moment gyroscopes to maintain its orientation towards the sun for solar power[23].

There are two main classes of gyroscopes when used for control. The first is a simple reaction wheel, in which a motor provides a torque between the craft and a massive wheel. The conservation of momentum concludes that both the craft and the wheel will rotate, proportionally to their moment of inertia in that axis, as Equation 2.2 shows. As the reaction wheel rotation rate increases, the craft rotation rate increases in the opposite direction.

$$T = \frac{dL}{dt} = I\frac{d\Omega}{dt} \tag{2.2}$$

Reaction wheels can provide a challenge in the presence of external forces, as corrective torque on the craft only comes from torque on the wheel. Thus, the wheel in theory may need to spin arbitrarily fast to correct for a constant disturbance torque on the craft. Spacecraft have additional actuators to deal with this that take advantage of external forces, allowing the wheels to despin.

The second class of gyroscope is the control-moment gyro. These control a craft through the phenomenon of precession, whereby a spinning object, when applied with a torque about an axis other than its axis of rotation, experiences a change of axis of rotation orthogonal to the axes of both the torque and the rotation. Alternatively, constraining the orthogonal axis results in torque being applied to the craft that is proportional to the "control moment," or the inertia of the wheel and its rotation speed, shown in Equation 2.3. Large spin speeds allow for weight and/or size (smaller I, same L) reduction while maintaining the ability to generate powerful torques.

$$T = \frac{dL}{dt} = I\omega\Omega \tag{2.3}$$

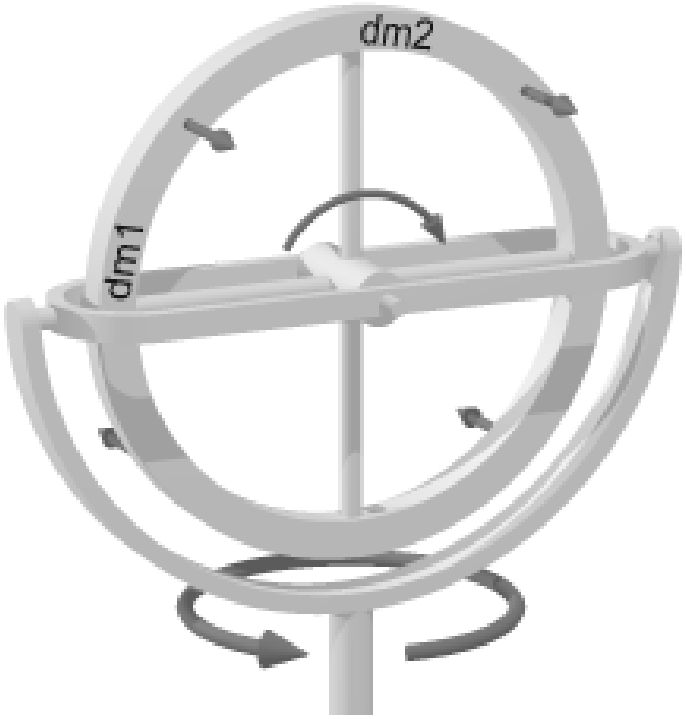

Figure 2-10: A control moment gyroscope uses the principle of gyroscopic precession to apply torques on a craft without the wind-up issues of a traditional reaction wheel. Note that if the horizontal axis of rotation is fixed, than the mass elements dm1 and dm2 cause a reaction force out of the page when forced to rotate about the vertical axis. When the same mass elements reach the bottom of the wheel, they cause a reaction force into the page. This produces a moment perpendicular to the spin axis and the forced rotation axis. [24]

Typically a spacecraft would have several of these, designed to control multiple axes and to provide redundancy. Unlike the reaction wheel, the moment of inertia of a control moment gyro changes as it is used, which means special care and extra degrees of freedom need to be built into a control system to prevent accidental alignment of the gyros with each other. This causes a condition known as "gimbal lock" where the gyros cannot be moved in the direction needed to cause the required torque. This also means that if a fixed torque generation axis is desired, multiple gyros are required.

Reaction wheels suffer from several qualities that obviate their use in the inspection robot. The robot may experience long-duration external forces due to variable lift, surface topology or dynamic coupling, causing saturation of the wheel. Additionally, the reaction torque they can provide is only as good as the motor providing it. Maxon, an industry leading producer of miniature, powerful DC motors, has no motors that could provide the required torque at stall with a mass less than 80g. The mass budget for the out-of-plane actuator is 50g, which eliminates a reaction wheel entirely. Additionally, multiple gyros are required since the axis of the reaction torque changes during use. Again, this does not align with the design goals of low weight and volume.

2.3.1.4 Attitude Adjustment Thrusters

An extremely common attitude adjustment system used by spacecraft is a Reaction Control System (RCS). Though the name can apply to any actuator used to adjust a spacecraft's attitude, it generally applies to propellant-fired thrusters used during infrequent maneuvers. For example, the Space Shuttle had a multitude of small thrusters along its body, designed to provide control torques and forces on the craft. These were versatile enough to provide rotation away from the booster stage and allow for a safe landing in the case of a mission abort in certain stages of the post-launch window, as well as provide small, measured periods of thrust to maintain altitude in orbit[25].

Typically, an RCS for astronautical use contains several orthogonal nozzles, placed in several locations on the perimeter of a spacecraft, similar to Figure 2-11. These

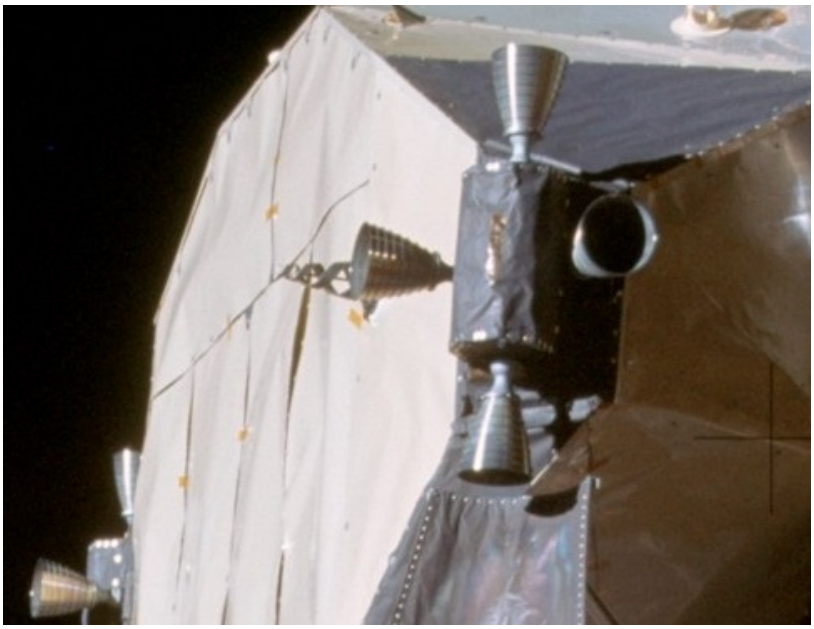

Figure 2-11: The sets of nozzles visible in both foreground and background on the Apollo Lunar Module provided fine control of body forces when fired in common, and body torques when fired differentially[26].

provide calibrated thrust vectors off the center of mass of the aircraft, and can work in tandem across the craft to generate a net force, and no net torque, or can work differentially to generate a torque and no net force. Depending on the maneuver required, some combination of the two effects provides the desired motion through space. Multiple valves control the amount, and/or the duration of propellant released during a "burn," or period of thrust generation. In comparison with gyroscopic attitude adjustment, the exhaust gases of the combustion leave the inertial system of the craft, so there is no conservation of momentum occurring that is of concern to the craft's future maneuverability.

In the case of the inspection robot, there would be plenty of fluid surrounding the robot to avoid using self-contained, expendable propellant. A single pump could be used to provide the flow necessary, and electronically-controlled valves could switch on specific thrusters in the configuration shown in Figure 2-12. Indeed, this was used for attitude control as well as the main thrusters on a similar tetherless underwater inspection robot, developed at MIT by Bhattacharyya and Asada[27]. Those authors cited challenges due to thrust misalignment with the center of mass (COM), from

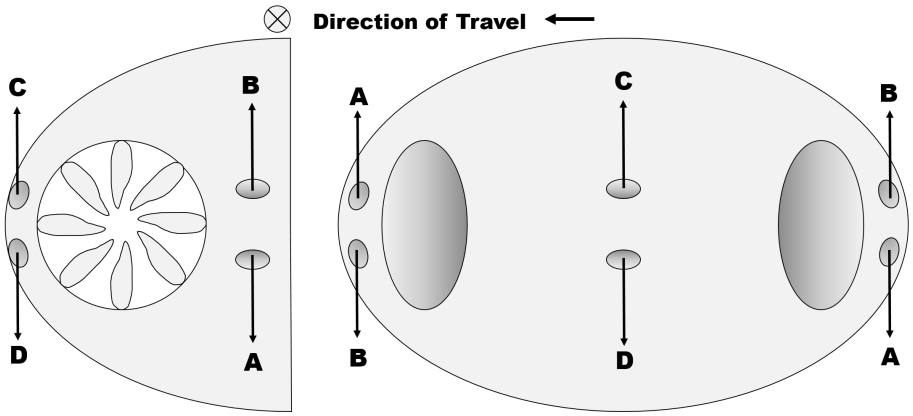

Figure 2-12: Concept drawing of an RCS applied to the inspection robot. If pairs A fire simultaneously, the robot pitches down, pairs B, pitch up. C and D would be paired to nozzles on the right-half of the robot, inducing roll.

both COM uncertainty and water jet characteristics.

The greatest challenge in using an RCS-inspired attitude adjustment thrust onboard the inspection robot would be size constraint. The robot discussed above was about the volume of a football, and could take advantage of commercially available pumps of sufficient power. The micro-pumps available, within the volume requirements of the robot, do not have sufficient thrust to provide attitude control[14].

2.3.1.5 Ailerons

Ailerons are a well-studied and widely implemented form of control actuation in the aeronautics and submarine sectors[28]. In essence, a small control surface at the trailing edge of a wing, such as that shown in Figure 2-13 is used to apply force on the craft as fluid flow impinges on the surface. By stowing the surface so it follows the same profile as the wing, no forces are generated. The surface usually may be actuated up, causing a downwards reaction force as it deflects the air flow upwards, or down, causing an upwards force.

An "aileron" is traditionally one such control surface mounted on the main wings, used differentially to roll the aircraft. From the outboard position on the wings, the forces generated can create large moments about the COM of the aircraft, resulting in effective roll control.

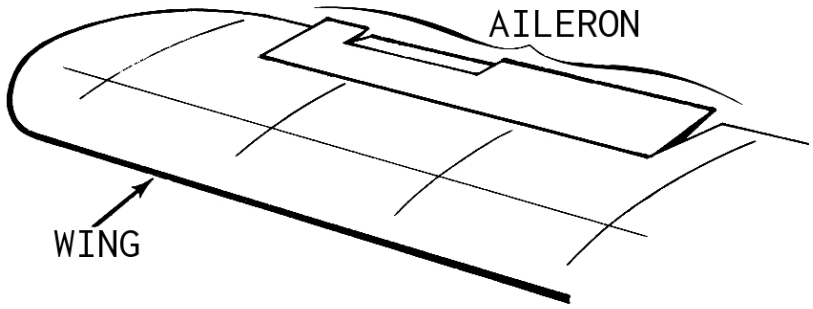

Figure 2-13: Typical use of an aileron on a wing of an airplane. The shown extended position would cause the plane to roll right if the matching aileron on the left wing was extended down. If this aileron were on a tail-mounted horizontal stabilizer, this position would pitch the plane up[29].

An "elevator" is another form of this same design at the tail of an aircraft. The elevator is mounted at the rear of a smaller wing, called a tailplane or horizontal stabilizer, which itself is at the rear extreme of the aircraft. From this position, forces on the elevator cause a large moment about the COM of the aircraft, resulting in a pitching motion.

These actuators are incredibly effective given their low weight and complexity because they leverage the existing fluid flow over the wings of the aircraft. Comparatively massive actuators provide the thrust for forward motion of the aircraft, and these smaller actuators redirect some portion of it for use in attitude control.

As it applies to the inspection robot, the scenarios are remarkably similar. The robot has its main thrusters, capable of producing high fluid-flows, and needs small, low-power actuators to control its attitude. Downsides are that the thrusters must be running to have attitude control, and the robot must translate relative to the surrounding fluid in order to adjust its pitch or roll. This also implies that the robot must translate horizontally in order to translate vertically.

However, these non-holonomic constraints are an acceptable tradeoff in exchange for the actuators' benefits. Notably, the robot would have full 6-DOF control in a moving flow, so could effectively park at a specific longitudinal position along a pipe and adjust its radial position and attitude to inspect some feature of the pipe wall.

Table 2.1: Pugh Chart of Additional Actuator Concepts

	Reaction Control System	Reaction Wheel	Control Moment Gyro	Thrust Vectoring	Additional Thrusters	Ailerons
Simple	0	+	-	-	0	+
Light	0	-	-	0	-	+
Small	0	-	-	-	-	+
Powerful	0	0	+	+	++	+

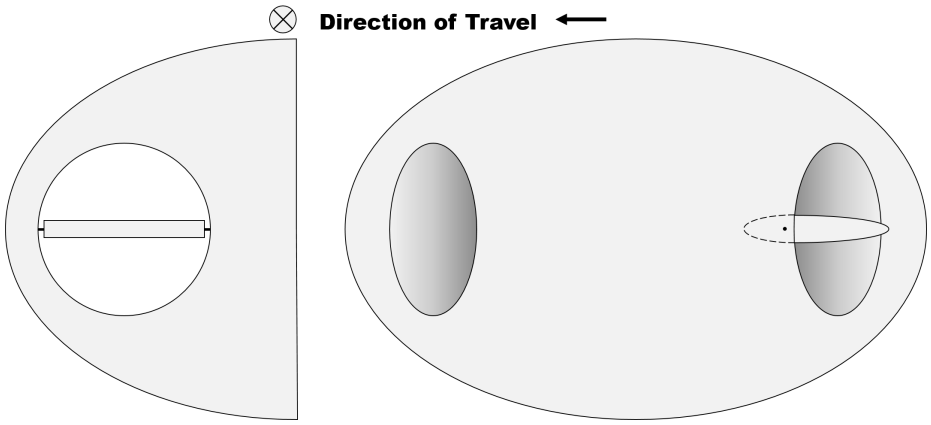

Figure 2-14: Initial concept drawing of aileron position in robot

2.3.2 Aileron Design

Based on the comparison of the proposed concepts, summarized in Table 2.1, some sort of aileron, utilizing the flow generated by the main thrusters, is a satisfactory design to add full controllability to the robot. Placing the ailerons within the ducts of the robot, as shown in Figure 2-14, maintains the same bounding volume, which is desirable to retain maneuverability within junction and potentially restricted pipes. An aileron, independently controlled, in each duct allows for pitch control via common-mode deflection, which also enables out-of-plane motion, and roll control via differential deflection.

A disadvantage of the aileron design is the loss of bidirectionality of the robot. Ailerons at the leading edge of the craft and leading edge of a wing result in highly unstable flight through the flow[30]. This will generally require the robot to pivot in order to change direction in the pipe. It should be noted, however, that in performing a pivoting maneuver, the robot will temporarily lose the ability to generate thrust

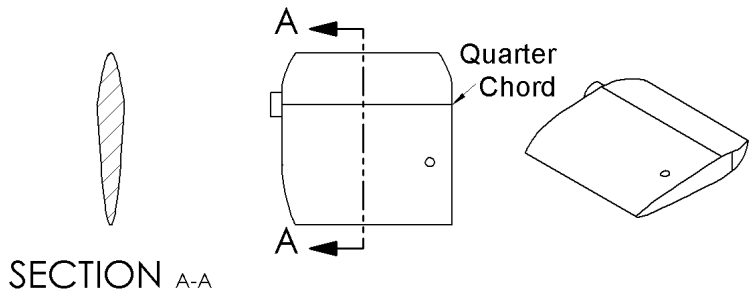

Figure 2-15: Aileron design showing cross-section and quarter-chord based pivot

along the longitudinal axis of the pipe. This may be of concern in pipes with high flow rates but may be mitigated by pivoting quickly. Pivoting about the yaw axis is known to be relatively fast due to the high thrust of the main thrusters. Alternatively, the highly unstable leading aileron could be utilized to perform a 180° pivot, potentially even more quickly, given an advanced controller.

The ailerons should generate no lift in the neutral position, and also minimize drag as it will act to decrease thrust. This is made possible by designing the aileron as a typical aerofoil with zero camber, such that the flow over and under the aileron at zero angle-of-attack is identical[31]. Drag is minimized in this configuration by minimizing the thickness of the aerofoil.

As shown in Equation 2.1 and Section 2.3.1.2, the robot is operating in a Reynolds number region lower than a typical aircraft. Literature for studies on aircraft aerofoil designs suggest that at low Reynolds numbers (starting roughly an order of magnitude above the scenario considered for this robot), the key factor driving aerofoil effectiveness at high angles of attack is the point at which the flow separates from the surface[32]. At the robot's lower Reynolds numbers, flow separation is less likely but can still be minimized by keeping the aerofoil thickness as small as possible, and placing the thickest section between a quarter and a third of the way back from the leading edge, as shown in Figure 2-15.

The aileron should be placed in the center of the flow for symmetric upwards and downwards pitching moments. Keeping the flow at an arbitrary angle up matched with the flow at equal angle down is critical for actuator linearity, simplifying control

analysis.

The pivot point of the aileron should be at its center of pressure, as this will prevent loading the aileron actuator with the lift forces. For a symmetric aerofoil, such as this one, the center of pressure remains at the quarter-chord distance, 25% of the aerofoil length back from the leading edge, for a large range of angles of attack[31].

2.3.3 Actuation and Control Lines

The aileron needs a means of actuation to provide the necessary control forces. Several methods were considered, with primary requirements being waterproofness, robustness, and actuator size.

The first method considered was a drive shaft powering the rotation of the aileron, sealed with a traditional doubly sealed bearing system, of which a "lip seal" setup is common. However, these setups commonly require a high wall to shaft diameter aspect ratio to ensure coaxial alignment of the shaft with the seals. Additionally, the robot lacks internal volume along the aileron axis of rotation, and fitting a motor would have required a 90° transmission to be feasible. The poor suitability and extra complexity required by this method obviated it as a candidate.

The thickness of the hull seemed the primary challenge in sealing any sort of shaft due to the aspect ratio. To reduce the wall to shaft diameter ratio, a smaller shaft and thicker wall are needed. To this end, another aeronautical inspired design can be utilized.

Prior to advanced hydraulic, then electric and drive-by-wire systems, control surfaces were actuated via control lines that were physically connected the pilot's foot pedals or joystick, as shown in Figure 2-16.

Also highlighted by Figure 2-16 is the use of antagonistic pairs of control lines, enabling bidirectional control of a surface using flexible cables in tension. By only using cables in tension, lightweight materials of a thin cross-section can be routed through the body of the craft, using idler pulleys as needed to change the direction of the cable with minimal friction. Antagonistic pairs stand in contrast to pusher rods which had to be stiff enough in compression to avoid buckling, requiring heavier,

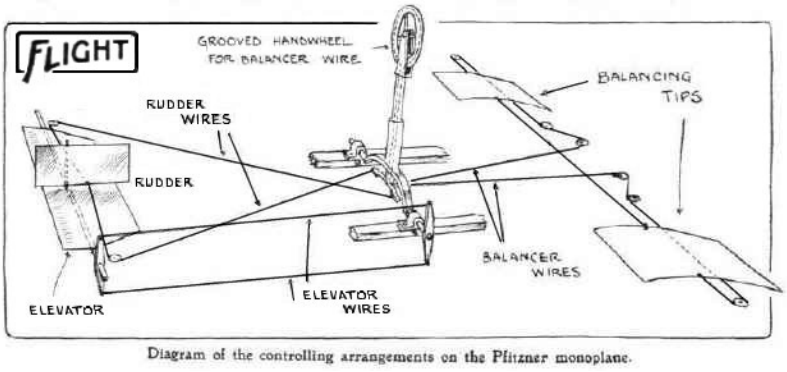

Figure 2-16: Control lines on small aircraft[33]

thicker, or more expensive materials.

2.3.3.1 Design

Control lines provide important benefits as a transmission system in mounting flexibility, light weight, and low occupied volume. The robot hull already provides a structure to use to route the cabling, so the remaining design elements required were motor mounts, bearing points, and the cable connection to the aileron.

For the motors, submicro hobby servos are commercially available and inexpensive. Models exist that are under 5g in mass, smaller than 25mm in length, and thinner than 10mm. Two such servos can fit in the robot, side by side, while leaving room for electronics and batteries. Two Power HD DSM44's were selected for the aileron actuators. The motors move the mass distribution, shown in Figure 2-17, to the rear of the robot, though with surplus buoyancy available, additional weights added up front can counteract this.

To transmit the torque from the motor to the control lines, a differential capstan was designed and mounted to the output shaft of the servos. This provides one simple mechanism which may retract the control lines on the side in tension, and release the lines on the side in extension, as shown in Figure 2-18. The control lines will be pre-tensioned, ensuring that there is no backlash in the motion and providing the necessary tension about the capstan to provide frictional force. The control lines were also tied to the capstan to ensure no relative slip would occur.

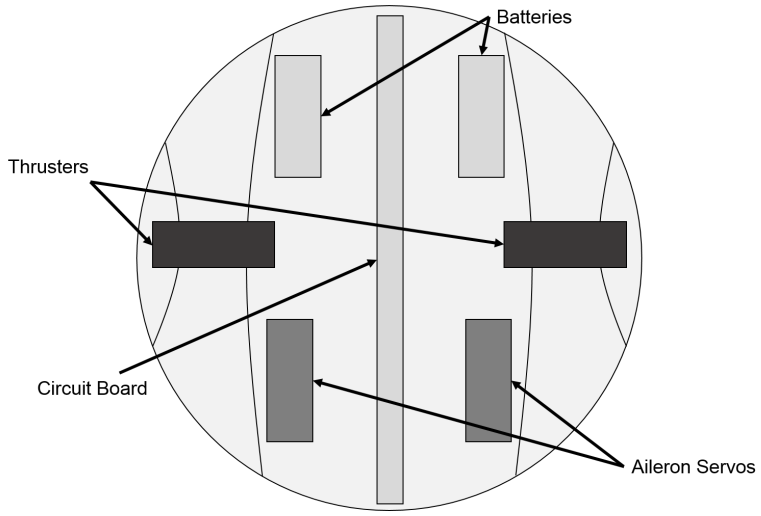

Figure 2-17: Heat-map of the relative mass distribution of the robot, with darker being more massive components. The predominantly metal components in the thrusters are the most massive part of the robot. There is spare buoyancy to correct the rearward mass distribution.

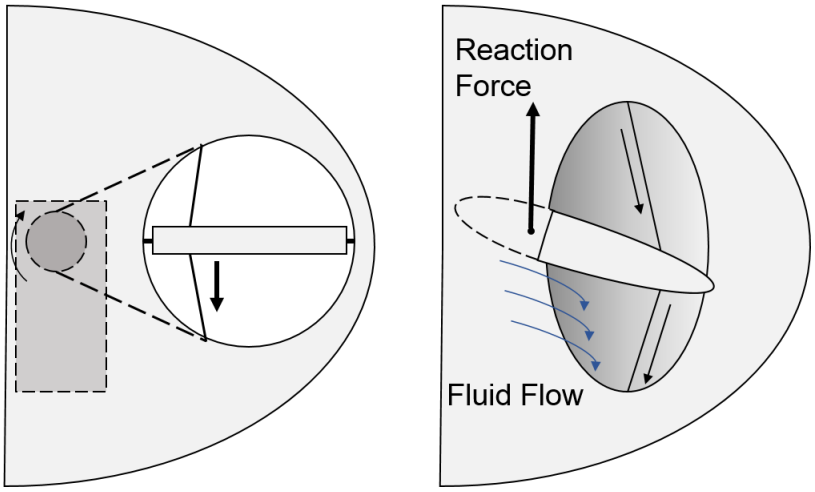

Figure 2-18: The control cables wrap around a capstan mounted to the output shaft of the servo (left). Servo rotation provides antagonistic cable movement (right), which assures bidirectional aileron control.

The properties required by the control lines are waterproofness, high extensional stiffness to prevent stretching, and strength so diameter may be minimized. These qualities are met by fishing line, particularly Spectra® brand woven polymer fiber[34]. Originally, fluorocarbon line had been tested but was too elastic under tension.

The capstan must have a diameter no larger than the body of the servo to ensure it would fit in the available space. Since hobby servos with built-in position control typically have very limited range, often ±90° or less, this limited the translational range of the control lines to $\pm\frac{\pi}{4}D_{capstan}$, or roughly ±4.7mm. The range of motion of the ailerons is tuned by adjusting the attachment point of the control lines. Moving the attachment point closer to the pivot point increases the angular range of motion of the aileron.

Finally, the tension of the line should be invariant with respect to the position of the ailerons. This is achieved to first order by making the exit point of the control lines from the hull nearly directly over the attachment point on the aileron. As the aileron rotates, the motion of the attachment point out of the axis of the control lines remains small. This is shown in Figure 2-18 as the control lines never stray far from their neutral angles through the rotation range of the aileron.

2.3.3.2 Waterproofing

A challenge with this design is that the junction where the control lines exit the hull must be waterproofed. The diameter of the control lines greatly reduces the minimum hole diameter compared to that of a drive shaft but further excludes traditional, commercially available sealing solutions. A novel means for creating a waterproof, pressure resistant seal that still allows the control lines to translate in and out of the hull is used instead.

The seal design is inspired by relatively modern cast-in-place sealing glands, typically used in the mobile electronics industry to make mobile phones water- and dust-proof. The seal is created to perfectly match complex mating surfaces by depositing an uncured polymer liquid onto one substrate, allowing time for the viscous fluid to conform to the surface, then curing it in place. The viscous fluid is not given

enough time to completely sag, resulting in a raised, compliant elastomer bead around a desired sealing surface. The lid or other components are then mounted onto the seal, and preload is applied to ensure the elastomer conforms to the surface topology of the lid.

For this research, the sealing surfaces are the small hull opening and the control line. In the first prototype used for this work, a smooth-sided fluorocarbon fishing line was selected for the control lines, due to its low stretch, abrasion resistance and low water absorption compared with monofilament lines. The smooth sides allow for excellent surface conformation with applied elastomer.

Initially, a silicon rubber, Smooth-On Mold Star 30, was selected to be the elastomer seal due to its compliance and ease of use. This two-part platinum-curing silicon rubber is considered archival due to its low deformation over time, temperature resilience, and chemical resistance (including water). The two parts were mixed. The cables were coated in mold-release agent, Smooth-On Ease Release 200, to prevent the silicon from adhering to them and add to a film of hydrophobic sealant to the seal-cable interface. The cables were then put into place, and the rubber poured into the hull around the cable. Rubber was also added to the outside of the hull, around the cable, and connected with the rubber on the interior. This formed a continuous seal around the cable, inside and out. Figure 2-19 shows the seal as implemented on the robot.

The original prototype, cast in a bowl with a punctured bottom, was filled with water and held the water for several days without signs of wicking or leaking. However, the rubber used was soft enough that applying tension parallel to the surface of the rubber deformed it enough to create a gap for water to flow through. The robot was designed with guides for the control lines to prevent lateral stretching of the seals. In implementing the control lines, it was discovered the tension on the lines was great enough to cause deformation of the fluorocarbon lines, so the lines were switched to High-Molecular Weight Polyethylene braided fiber lines, namely the ones sold under the Spectra® brand. These lines are much stiffer, and showed no noticeable extension or deformation in use on the robot.

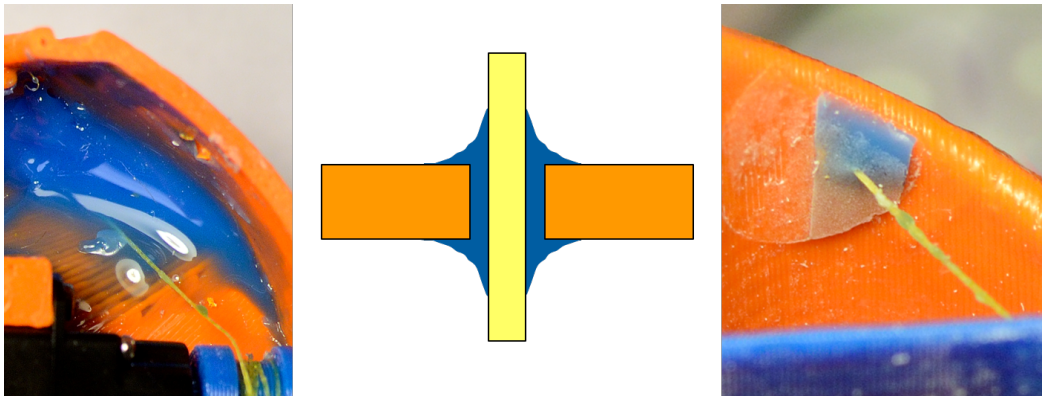

Figure 2-19: The view of the gasket formed in place over the control line from the inside (left) and outside (right) of the robot. The rubber seal connects between the inside and outside of the robot for mechanical stability as the control line moves in and out (center).

Some stronger control line should be found with a smooth outer bore, such that the seal can clamp the entirety of the surface rather than the pockmarked braided surface of the Spectra. Silicon rubber is not the best choice for this application, as it resists bonding to surfaces, so any mechanical adherence that is achieved can easily be defeated in soft rubbers. Urethane rubbers adhere strongly to the plastic components used in the hull, so would provide chemical adhesion to seal that surface. The same release agent can be used on the control line to prevent the rubber from adhering to it. However, depending on the needed travel in the control line, an alternative seal exists. A soft rubber could be adhered both to the hull and to the control line, creating a diaphragm that would be chemically sealed to both components, and would move in and out with the control line to allow its function while retaining seal. This can be explored further in future work.

2.4 Conclusion

Several possible designs for out-of-plane actuation were considered. Underwater drones frequently make use of technologies developed in relation to the aerospace industry, and a priori comparisons are made based on design criteria and performance analyses in those applications. Adjustments for fluid-flow differences are ap-

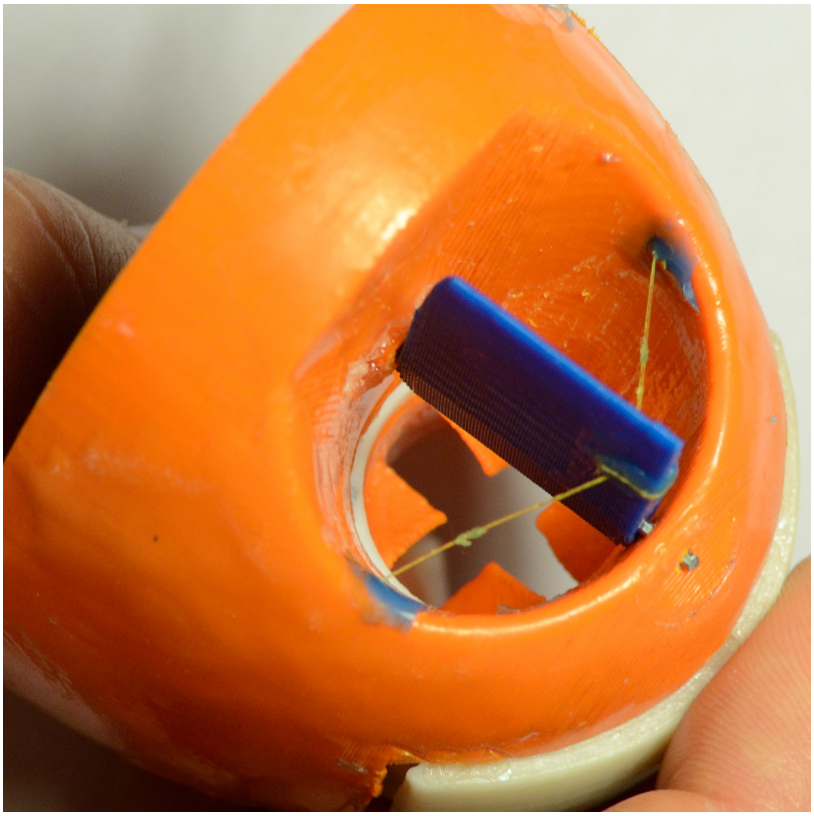

Figure 2-20: Design of aileron and mount. The angle of the aileron (shown angled up) can be adjusted by the cables the run to the inside of the robot, in order to deflect the thrust from the propeller

proximated by considering the Reynolds number variations between pipeline fluid flow and altitude plane flight, with the conclusion that effects already dominated by viscosity in aerospace applications will continue to be so dominated in water pipelines.

Ultimately, an aileron control surface is selected due to low power, mass and volume. Its design utility is primarily in controlling the powerful actuators that already exist on the robot, rather than trying to add more high-power actuators with unreasonable power density specifications.

A control line transmission is developed to power the aileron. Extremely lightweight, high-tensile strength polymer thread is selected to provide the transmission, minimizing the hole diameter required in the hull. This minimizes the amount of surface that needs to be sealed, presenting a smaller failure surface.

Final design weight was 155.4g, and given the 175g equivalent displacement of the

hull, the robot is slightly buoyant. Weights can be added to give neutral buoyancy, or better yet, additional battery capacity could be added.

Additional work is needed to evaluate the form-in-place sealing system. Future work may also consider designs not requiring a penetrative transmission. There is value in providing a hull design with no holes, such that it can be guaranteed to stay sealed under reasonable use. Additionally, non-smooth features, or features that move from the inside to the outside of the craft, can harbor bacteria. This makes it difficult to prove safe operation in a drinking water supply. A solution like the diaphragm seal proposed may offer the best seal while retaining the features of a closed hull.

Chapter 3

Electronic Design

The demanding conditions of pipeline navigation require a dedicated controller in the loop to manage the actuators on the robot. Additionally, if the robot is to have any autonomy, significant processing power is required to interpret sensor streams and maintain knowledge of location in a potentially massive pipeline network.

As many pipelines employ metallic pipes, RF-based wireless communication is assumed to be unavailable in general navigation. Additionally, maintaining any form of wireless communication across the entirety of a municipality's water system would be difficult, likely requiring many repeaters that would be infeasible to retrofit.

Wired communication is similarly undesirable, as the robot would be required to drag around a massive and lengthy tether, providing unnecessary mass and drag forces. It is also desirable to be able to move anywhere in the system without worrying about creating loops in the tether, or having to keep track of the tether location.

Instead, the robot may navigate through the pipeline autonomously, under its own carried power. As needed, either for urgent repair or due to a full memory, the robot could go to one or several data uplink points in the network. Here, a relay would allow the robot to upload its data or specific alerts, and operators could observe robot health and state. Optionally, these locations could also function as insertion and extraction points for the robots, allowing physical repair, sanitation, or any other interaction needed.

An embedded controller is needed to perform all required computation and control

on-board the robot, log data as needed, and communicate with the uplink point. This chapter describes the design and methods employed in fabricating such a controller. Appendix B reproduces the circuit schematics developed in this work.

3.1 Previous Work

Previously developed by You Wu[14], the robot utilized two Arduinos, in a processor-coprocessor arrangement, to read IMU data, control the motors, and communicate with a controller on a host computer in LabView. This setup utilized all commercially-available breakout components for ease of testing but at the loss of space efficiency. No room was left available for additional actuators for out-of-plan motion. Additionally, the 8-bit, 16MHz processors on board the Arduinos are not capable of running high update-rate non-linear controllers as is needed in the full dynamical control of the robot. To this end, an integrated, space-efficient, powerful embedded processing and control board is developed in this work.

3.2 Requirements

The design requirements for the inspection robot are multi-disciplinary in nature. Special attention must be paid to control systems, memory use, power consumption and power output.

In order for the robot to perform at least as well in maneuverability as the version presented in [14], it must have on-board brushless motor controllers capable of driving the motors at 2A continuously. The previous work on motor design left no mounting points for hall-effect sensors to indicate commutation position, requiring the motor driver to be capable of sensorless operation. These controllers use the back EMF generated by the rotor moving relative to the windings in the stator to detect where in the commutation sequence the rotor is. For control feedback, the motors should also have a tachometer output to determine thrust.

As the robot must be designed for long-term submerged operation, the ability to

enter a low-power state is necessary. The robot must remain sealed while underwater, so battery drain during rest is a major concern. It may be able to park at a charger or find some other means to keep its battery charged (see Section 3.11) but should still optimize its power usage in an inactive state. This will generally require the ability to shut power off to peripheral devices, to slow down the processor or put it to sleep, and to charge the batteries from an external power source.

As a prototype for the development of an inspection robot, it is desirable to provide comprehensive logging capability. This includes high-fidelity sensor and kinematic information during maneuvering to test and validate dynamic models and controllers. This will require an external non-volatile storage device capable of a high data rate, and a microcontroller (MCU) that supports a high data rate with the external memory. Additionally, an MCU that has a Direct Memory Access Controller (DMA) and a memory controller that supports DMA copies will perform vastly better. A DMA is essentially a second processor, whose only job is to read and write memory. In this instance, one could be configured to watch the IMU data structure, for example, and copy its contents to memory whenever they change. This provides major improvements in system performance, as the Central Processing Unit (CPU) is free to continue executing programs during this time rather than waiting for the memory transfer to complete.

The robot must have sufficient processing power to run position estimation and control algorithms at a high enough rate that it can stabilize its position in the pipe. A Floating-Point Processing Unit (FPU) will greatly increase the speed of floating-point math, which simplifies coding practices and data-types. This should include peripheral sensing elements like the IMU for best results. A goal control loop frequency of 200Hz was used for the following design. Since the robot's body and the pipeline features have roughly the same size, the robot must have several controller ticks within this characteristic length of 100mm. At a speed relative to the pipe of 1m/s, the desired max speed of the robot, a controller at 100Hz has 10 controller cycles within the characteristic length. As will be discussed in Section 3.8, the IMU chosen for the board runs easily at 200Hz, which provides double the controller rate

of this rough minimum-rate calculation.

3.3 Processing

The MCU chosen for the robot is an STM32F446VE produced by ST Microelectronics. It is a 32-bit ARM® Cortex®-M4 processor with a built-in single-point precision FPU. It can run up to 180MHz, and was configured for 72MHz operation in this work. Details of how the MCU is connected can be seen in Figure B-2 and Figure B-1.

It has several low-power modes with increasing levels of peripheral shutdown, including everything disabled except the interrupt subsystem and the real-time clock (RTC) and the processor halted. This means it would be compatible with shutting down the entire robot and waiting for a communications event or IMU interrupt to wake it. This MCU is designed to have one input voltage, and has internal regulation to provide the core with a lower voltage (for reduced power consumption). The fully-active current consumption at 72MHz with all peripherals enabled (but neglecting GPIO output) is under 50mA. Supplying the MCU with a higher supply voltage increases oscillator stability and improves digital output performance, at the cost of high power-consumption.

This processor also has a Quad-SPI flash memory peripheral, which can run at very high data rates; over 80MB/s dependent on the hardware it is connected to. In this application the processor achieves instantaneous data transfer rates of 11.8MB/s but is limited by the external flash to a time-averaged 170.7kB/s.

The advanced nature of this microcontroller is highlighted through its instruction and data cache, along with its prefetch controller. Though the on-board program flash memory in the chip has high latency relative to the processor speed, the memory controller actively reads ahead in memory, storing the next set of instructions to be executed. This allows the processor, in most cases, to run at full speed without having to wait for the flash latency time. Additionally, small loops may be loaded entirely into the cache, allowing full-speed execution with no memory interaction.

This allows the CPU to run instructions at its set frequency most of the time, which is much higher than traditional 1 instruction-per-clock-cycle CPUs like the Arduino.

The processor also provides a DMA controller. Though the internal memory interface can only allow one memory-bus "master" at a time (typically the CPU), the DMA takes advantage of the memory-access time saved by the instruction cache to automatically read and write sequential addresses from memory to memory, memory to peripheral or peripheral to memory. This is specifically useful in datalogging, in the memory to peripheral mode, as it can automatically coordinate with the Quad SPI memory peripheral to write data to the external flash storage. The CPU is not involved and can continue executing the program while flash writing occurs. This vastly improves the processor throughput in the case of a constantly-logging controller, as there is a lot of data to write every control-loop cycle.

Finally, the physical size of this processor is excellent, coming in pin counts ranging from 64 to 144 pins and several form factors. For ease of soldering, the largest form factor was chosen, at 14mmx14mm for the 100-pin version but an 81-pin version is available at just 3.7mmx3.8mm.

3.4 Power Subsystem

This subsystem is comprised of two main functional groups; charging and voltage regulation. Testing the robot will be facilitated by minimizing how frequently it must be opened. This will prevent physical wear on the sealing surfaces, extending their lifetime. Additionally, accidentally applying power incorrectly, i.e. through plugging the batteries in backwards, can be effectively avoided if the batteries are always connected. An integrated charging system was implemented as a solution.

Note that the following section will refer to a Lithium-Polymer battery consisting of one or more cells in parallel as a 1s LiPo. This is to be consistent with calling a Lithium-Polymer battery consisting of 2 series stacks, where a stack may be one or more cells in parallel, a 2s LiPo. This follows conventional LiPo naming schemes, which inform the user of the properties of a pack as a whole, rather than its particular

make-up. LiPo cells can vary greatly in electrical capacity, and may be stacked in parallel to generate identical capacity with an arbitrary number of cells. Since in this section we are more concerned with the nominal voltage of the battery, the number of stacks put in series is the focus of the notation.

The MCU chosen in Section 3.3 cannot directly take a connection to a LiPo battery for its power supply. A 1s LiPo battery has a maximum (fully charged) voltage of 4.2V, while the MCU has a maximum allowable input voltage of 3.6V. Clearly, the power supply to the MCU needs to be regulated to a lower voltage. The actuators, however, benefit from a higher voltage supply, which helps maintain torque output at high rotor speed/back EMF. The previous research by You Wu showed that the target speed of 1m/s was nearly achieved with a 2s battery powering the motors, so this configuration was used here as well.

In this section, we discuss a design for a flexible, balancing, 5V-chargeable, 2s LiPo battery configuration, using software switching for smart voltage-level conversion.

3.4.1 Charging

The batteries must stay connected to the controller, and the controller will therefore remain powered on indefinitely. The batteries would die relatively rapidly in this scenario, unless provisions to charge them are made.

Traditional hobby-grade battery chargers cannot be programmatically enabled/disabled, rendering them unsuitable for this system. Since the robot could be connected to the charger while idle for long durations, the charger needs to be able to restart charging the batteries if they drain, without human input.

A built-in charge management design was selected instead, so the MCU would have full control over when to start and stop charging. Commercially available chip-level solutions focus on complex chargers for generic, multi-cell LiPo batteries but the topology could be vastly simplified in the case of a 2s battery.

The charge management system must charge the cells of the 2s battery in a balanced manner. Battery balancing means charging the cells to full capacity individually. This is necessary since each cell's capacity will vary slightly, meaning that

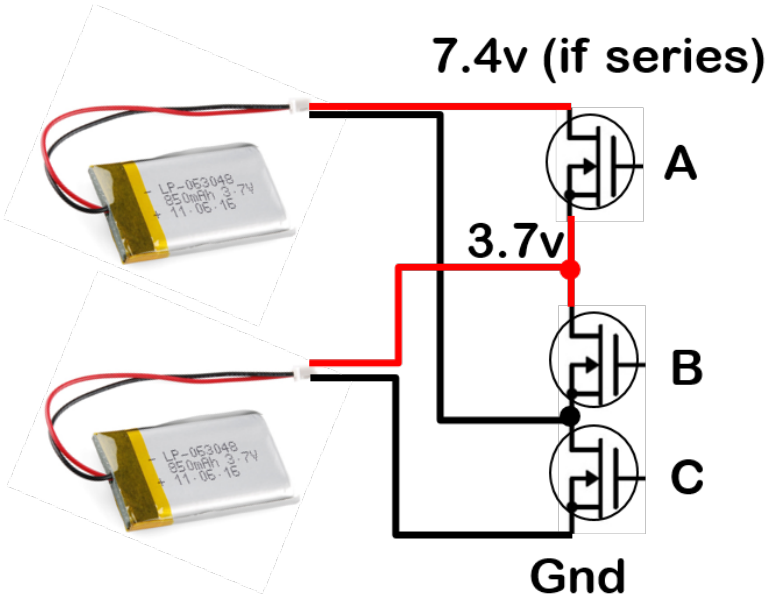

Figure 3-1: The batteries are capable of being connected in either series or parallel, in Run mode or Charge mode respectively. The MOSFET switches follow the state table shown in Table 3.4.1.

Table 3.1: Battery Switching State Table

Switch	Series	Parallel
A	OFF	ON
B	ON	OFF
C	OFF	ON

one cell will drain first, and the other cell will subsequently reach a charged state first. Since the system will be connected for many charge/discharge cycles, imbalance will cascade into the lower-capacity cell being relatively discharged even when the higher-capacity cell is fully charged, significantly reducing the safe run-time of the robot.

To provide balanced charging, a novel approach was used. Rather than provide a complex solution to float the voltage at the midpoint of the two cells, or use power resistors to dump excess charge on the higher-capacity cell, the design charges the two batteries in parallel, then switches them to series when the robot is ready to run. The arrangement of MOSFET switches with the batteries is shown in Figure 3-1.

Connecting two cells in parallel must be done carefully, under controlled condi-

tions. The same factors causing battery imbalance can lead to voltage differences between "fully charged" cells, resulting in differing voltages between them. Parallel connection can cause rapid discharge of one battery into another, and must be guarded against. In this application, the batteries will be rebalanced every charge cycle, and thus won't reach a dangerous level of imbalance. However, the batteries used have protection circuits built in to detect overload conditions, for safety in the event of a failure.

Charging the batteries in parallel has another benefit besides balancing; it reduces the voltage required in the charging supply. There are many space-optimized designs for mobile devices that provide a USB 5V-compatible charging interface for a 1s LiPo battery. When the batteries switch into parallel configuration, they both may be charged from a USB cable. This is convenient for the development of the platform, where the microcontroller can be programmed via USB, tested, debugged, and charged all through a single cable.

3.4.2 Voltage Supply

Several supply voltages are required across the controller board.

The thrusters, demanding high peak currents, are best served being connected directly to the batteries. LiPo batteries are highly-regarded for their low ESR, which translates to providing a steady voltage even at high current draw.

The hobby servos controlling the ailerons cannot, however, be connected directly to a 2s LiPo due to maximum input voltage limits of between 6-7V. These servos have a much lower current demand than the thrusters, so a small-footprint buck-converter system was selected. This is comprised of a TPS62160, and a small-footprint 2.2uH inductor, configured to output 6v. The regulator is capable of up to a 1A output, while the current draw of the two hobby servos was estimated to be about 250mA. The regulator design as implemented occupied 38mm^2 on one side of the board.

As discussed in Sections 3.3 and 3.4, the microcontroller needed a regulated supply. Higher supply voltages improve oscillator stability, allowing the MCU to run at higher speeds. To ensure proper operation at 72MHz, as well as good compatibility with

commercially available sensors and peripherals, a supply voltage for the MCU was chosen of 3.3V. To prevent noise from peripherals affecting the operation of the MCU, in particular its analog domain, the MCU has a small, dedicated supply.

With the MCU power designed to be an independent supply, the rest of the peripherals need a power supply. A second 3.3V regulator is added to supply power to the IMU, external flash memory, motor controllers' digital signal supply and the radio transceiver. This supply has a digital enable/disable pin to allow the entire peripheral domain to be turned off in low-power mode.

3.4.3 Integration

A challenge posed by these requirements is the safe handling of the battery parallel-series transition. This means more than just safely connecting the two cells in parallel; there is a problematic transition to a 1s configuration where, with fully drained cells, the voltage regulator for the MCU may not have the minimum voltage overhead it needs to continue providing 3.3V.

Additionally, at battery plug-in, the regulators are entirely powered down. The desired power-on-reset state of the batteries is the 1s configuration so they may charge if they are extremely low, and to keep the robot running as long as possible in extreme low-battery conditions. In this state, the MCU power supply is disabled, and has never been on, so the MCU has no power supply at all.

To resolve the aforementioned issues, the power-domain is provided with an "emergency" power shunt diode, shown in Figure 3-2, that brings the MCU power rails up to the minimum working voltage of the MCU until the batteries can supply a steady 3.3V rail. By choosing a power Zener diode with an appropriate Zener voltage drop, the battery midpoint, which is always at the 1s voltage, can provide safe power to the MCU through the whole battery range, at the expense of current handling and efficiency. The MCU should only be in a minimal power state when relying on Zener diode power. A Zener diode is chosen for this due to its stable voltage drop characteristic compared to a normal diode in either forward or reverse bias conditions.

Since a Zener diode still has a forward voltage drop, and the battery should never

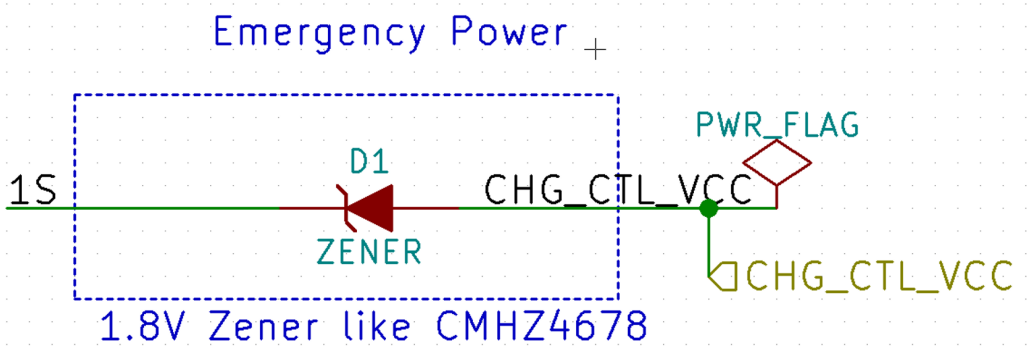

Figure 3-2: D1 is the emergency power shunt diode, responsible for providing power to the MCU during initial battery plug-in and during extreme low-battery events. 1S is a direct connection to the lower LiPo battery; CHG_CTL_VCC is the MCU power rail.

Table 3.2: Power-On Defaults for Power Subsystem

Subsystem	LiPo	MCU	Servo	Aux	Charger	Zener Shunt
Power-On	Par	On	Off	Off	Off	Conducting

get below 3.3V, the system is guaranteed not to reverse conduct when the regulator is running, which would short the battery through the regulator.

All regulators and the MOSFETs providing the switching for the 1s to 2s battery conversion are controlled by the MCU. At power-up, their default states are selected by pull-up and pull-down resistors, as shown in Table 3.2. A comprehensive state transition controller monitors the battery status and charger state to safely provide control signals.

3.5 Communications

An integrated radio-frequency (RF) wireless-communication chip, the HUM-900-PRO, is used to provide bi-directional communication between the robot and a base-station, as shown in Figure 3-3.

Water is a strong attenuator for microwave-band signals, with decreased attenuation at lower frequency. The robot is designed to operate autonomously except to upload data, so under a meter wireless range is not a concern. Still, the greater the range the more convenient for testing, so a 900MHz system was selected.

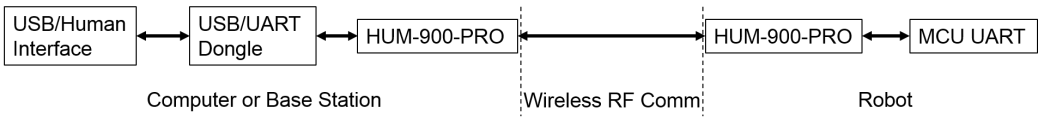

Figure 3-3: The human interface design in Python communicates with the robot via 900MHz radio transceivers.

It was not a design requirement to have high-throughput data transmission, and the common "high-speed" choice is a data-rate of 115200 baud (raw bits-per-second). This rate is supported by most common radios.

Due to the complex nature of transmitting and receiving RF signals, modulating data into the carrier wave and building a full data-link layer on top of the physical layer, a commercially-available pre-packaged solution is desirable. Several manufacturers offer these products, such as the Xbee by Digi International, but only the HumPRO offered a 900MHz transceiver in a chip-scale, board-on-board package. As volume and circuit board area are the top design constraints, this was the radio solution chosen.

3.6 Data Logging

To capture and evaluate robot dynamics during testing, and to store and upload inspection information during use, the robot needed long-term data storage. Logging robot dynamics is the operation requiring the higher data-rate, so it was used to determine the requirements for a memory solution.

As discussed in Section 3.2, a minimum control-loop frequency of 200Hz was selected. The floating-point unit on-board the MCU can only manipulate single-point precision numbers efficiently, so a single "value" is considered to be 4 bytes long. We consider the data-rate required to store the estimated 6-DOF pose of the robot (ideally with rotation as a quaternion), the rotation rate and accelerations from the IMU, prescribed thruster and aileron commands and a time-stamp. That represents 16 values, or 64 bytes, though the thruster and aileron commands could be compressed to 2 bytes each if additional data logging were needed.

Though the robot would have power continuously during operation, it was decided the memory should be non-volatile. This allows the memory to be powered on the auxiliary power rail, without losing data any time the robot goes into a low power mode. Additionally, during testing, the batteries may be removed before data can be recovered. No loss will occur in these scenarios with non-volatile memory. The small volume requirement, no hard requirement on capacity, and physically dynamic environment obviate the use of a mechanical storage solution. The MCU has enough RAM to buffer data in exchange for a higher data-rate, so a Flash NAND device was selected over Flash NOR or EEPROM.

On an early prototype, a parallel addressing flash memory was used, but the 24 address lines required the 144-pin version of the MCU, which is considerably larger. To save circuit board area, a Quad-SPI interface was selected which allows the smaller 100-pin count version of the MCU, while retaining a high transfer data-rate compared with SPI or I2C interfaces. The memory chip used in the final revision was the SST26VF064B by Microchip. This chip provides 64Mbit of capacity, split into 256 byte pages.

Since a datalogging packet is 64 bytes, 4 packets are buffered to fill a page. Flash NAND devices allow selecting a page to write, read in the data, then require a wait time to finish programming the selected page. Therefore, in software, a page is buffering while another page is being written. Once a page is full, it is immediately swapped out with an empty page, and written to hardware. This allows the MCU to perform the fastest data write rate possible on the memory chip. The datasheet for the SST26VF064B specifies a maximum wait time of 1.5ms after writing a full page of 256 bytes. The time spent sending the data to write the page is negligible compared to the write-wait time but even at 2ms worst case timing, we could write 500 pages per second, which would equal 2000Hz in control-loop log packets with the current design. This data rate exceeds current need, allowing for significant future growth in logging capability.

3.7 Motor Control

As discussed in Section 3.2, the previous work determined the motor design to draw 2A continuous under demanding conditions. This lies outside the range of most single-chip brushless, sensorless motor driver solutions, which are typically geared towards disc-drive motors and small cooling fans.

The A4960 by Allegro Microsystems is a sensorless, brushless motor driver designed for fluid pumps. It uses a SPI interface, which is convenient since adding a second motor driver only requires one more pin for the digital control interface. The driver includes a built-in charge pump to drive external MOSFETs with 15V, the typical guaranteed full-on gate voltage of power MOSFETs. It provides an adjustable voltage digital interface, allowing 3.3V interfacing with the MCU while the motor supply is connected directly to the 7.4V nominal battery.

The half-bridge drive is designed with bootstrap capacitors, allowing it to drive N-channel MOSFETs from its single, 15V charge pump. N-channel devices are typically smaller for a given power rating than P-channel, due to the higher mobility of electrons as a majority charge carrier compared with holes. The MOSFETs are added externally, allowing for design freedom in choosing the $R_{ds\text{-}on}$ and package. From the powerful gate drive voltage, the MOSFETs would be very near the minimum $R_{ds\text{-}on}$. Additionally, the dedicated gate-drive circuitry allows for high gate currents, making high gate-capacitance devices feasible to drive at PWM frequencies. Allowing high gate-capacitance means power devices with larger gates can be used, maximizing channel size which minimizes $R_{ds\text{-}on}$. In addition to the lower power dissipated from a lower $R_{ds\text{-}on}$, the larger channel used on the chip also spreads the power dissipation over more of the chip's area, decreasing the thermal resistance to ambient and to the circuit board.

The N-channel MOSFETs selected were PMV45EN2 produced by Nexperia (formerly NXP Semiconductor). These MOSFETs have a 30V maximum V_{ds} and can handle 4A continuously when mounted on a radiative copper pour on the PCB. The combined area of the motor driver and the 6 external MOSFETs provided comparable

heat-sinking surface area to the reference design in the datasheet, and no significant heating of the MOSFETs was observed during testing.

The A4960 motor driver provides a velocity-related output, though it cannot be called a tachometer. The output is a pulse train that follows the commutation state of the motor driver. The output goes high when the measured back-EMF on the non-active winding crosses 0. The output goes low again once the motor driver determines the commutation state should proceed to the next state, using an internal adaptive timing system. By tracking the time between high-going edges, an estimate for velocity can be obtained. This estimate will be noisy since the back-EMF zero-crossing detection itself is subjected to a noisy signal. There also is no velocity estimate or control during the first few commutations, as the motor driver must attempt to open-loop rotate the motor until enough back-EMF is generated to detect. This pitfall exists for all sensorless brushless motor drivers.

3.8 Sensors

In this work, a 9-DOF IMU was added to the robot platform. By integrating a dedicated, chip-level IMU, the system gains several performance benefits. Data-rate can be pushed to the maximum available for the chip, without relying on intermediate data format conversion. Hardware interrupts can be generated to offload timing and repetitive tasks; most critically, waiting for the incoming data to be ready. The exact orientation of the IMU can be adjusted for best performance and easiest manipulation; by placing the gravity vector along a specific axis for example. Lastly, no bulky, additional PCB needs to be included in the interior, taking up valuable space with mounting and inefficient "break-out" routing.

The IMU utilized is an MPU-9250 made by Invensense. This IMU has been commonly used in Android phones, and Invensense provides libraries to assist firmware development. The IMU provides 3-axis acceleration data, 3-axis rate gyroscopic data, and 3-axis magnetometer data. It includes an automated self-test that doubles as a calibration routine, providing constants to adjust for manufacturing variability. Also

included is an internal, adjustable, low-pass filter to reduce signal noise. This is especially convenient in this application due to the high effective mass of the robot due to viscous and added-mass forces.

Most usefully, this IMU has an on-board Digital Motion Processing™ (DMP™) co-processor that performs sensor fusion. Invensense provides a binary blob as a ready-made firmware to upload to the co-processor in order to perform this sensor fusion, making pose estimates available to the main processor without requiring any further computation. It must be noted, however, that the binary blob is not transparent, so the algorithm and accuracy of the fusion is not known and needs to be evaluated.

3.9 In-System Programming

As this robotics platform is intended to be a prototyping device, altering the firmware is of critical importance to the development cycle. Making this step as simple and fast as possible is desirable.

A general programming procedure for any embedded device project is: code, compile, upload, debug. Coding is the process of writing the software, and is covered in Section 4.

Compiling, or compilation, is producing the binary file to be programmed into the MCU memory for execution. For the STM32F4 series of MCUs, the popular, free/libre software tool GCC is available to compile C/C++ code into ARM Cortex-M4 binary data.

To upload the generated binary to the processor, both software and hardware is required. ST Microelectonics produces a development board for the STM32F4 series of processors. A feature of this board is that it contains an on-board programmer that can connect to a computer via USB, and can selectively program the MCU on the development board, or one connected externally. The programmer connects to the robot via a 5-wire interface. These development boards are widely available and inexpensive, as well as supported by free/libre software.

The software that supports this programmer is called *stlink* and is available on

Github with a BSD license at `https://github.com/texane/stlink`. *stlink* provides a TCP connection interface for higher-level software to send commands to the MCU by translating those commands into USB packets which are sent to the programmer which communicates with the Debug peripheral in the MCU. This allows the second software utility, also a free/libre program, GDB, to perform the upload of the program binary to the MCU.

GDB also functions as the debugger during program execution, if connected. Debugging is a critical function of any software development, allowing inspection of variable and register values, manual modification of registers, pausing and restarting of program execution, and many more beneficial features. These features allow the program to test code comprehensively, as well as find code paths causing issues if they arise. GDB connects to *stlink* to control the processor, and perform other actions on the MCU.

3.10 Assembly

The PCB to contain all the above features was designed in Kicad, a free/libre circuit design package under development by CERN. The PCB was fabricated as a two-layer board by OSH Park, with high-quality ENIG pad coatings and good solder resist. The size requirements of the board required minimal component area, so all components were selected as surface mount except wire connections to the batteries or motors. The components were hand soldered using both a soldering iron and a hot-air station. In particular, the MPU-9250 IMU has no exposed leads, so must be soldered with hot-air or in a reflow oven. The final board is shown in Figures 3-6 and 3-7.

3.11 Proposed Regenerative Charging for Unlimited Inspection Time

A noted limitation of the current robot design is the direct conflict between limited volume and long-range operation. The batteries currently used are not capable of

Table 3.3: Values Relating to Range

v_{max}	Current @ v_{max}	Battery Capacity	Runtime @ v_{max}	D_{max}
0.3m/s	2A (total)	300mAh	405s	162m

operating continuously for long enough to travel tens of kilometers of pipeline; a requirement for the robot to travel the distance between sparse, or singular, charging stations. Though more custom-shaped batteries are an option to add more capacity without modifying the internal volume of the robot, this solution could do no more than to increase capacity by a factor of 10 by using all remaining volume in the hull. With a currently estimated range of 200m (see Table 3.11) the best a battery upgrade could do would be to give the robot on the order of 2km of range.

A battery upgrade is necessary, but can be radically augmented by giving the robot the ability to charge passively, in place. This would be made possible through modifications to the motor driver circuitry, namely in changing the way the half-bridges are connected. In the regenerative charging mode, if an external torque rotates the propeller, the voltage generated through the windings is captured, rather than burned as heat in the half-bridge MOSFETs or carelessly recirculated to the battery.

Since the voltage generated by the motor is still sinusoidal, zero-centered, and spread across three phases, a zero-crossing detection scheme can be implemented to determine the location of rotor in the commutation sequence. This is exactly the same as the back-EMF commutation used in the motor driver. As the "sense" winding is crossing 0V, the other two windings are generating a maximal voltage between them due to their own induced back-EMF.

By switching the correct MOSFETs on, a low-resistance path can enable current to flow from source to drain, increasing the voltage at the positive motor rail relative to the negative rail. In the regenerative charging mode, the motor rails should be disconnected from the battery, and instead connected to a voltage regulation circuit, tied to the battery charging circuit. This topology is illustrated in abbreviated form in Figure 3-4. Though less efficient than a direct connection to the battery charger from a power supply, this method could allow the robot to run indefinitely, without

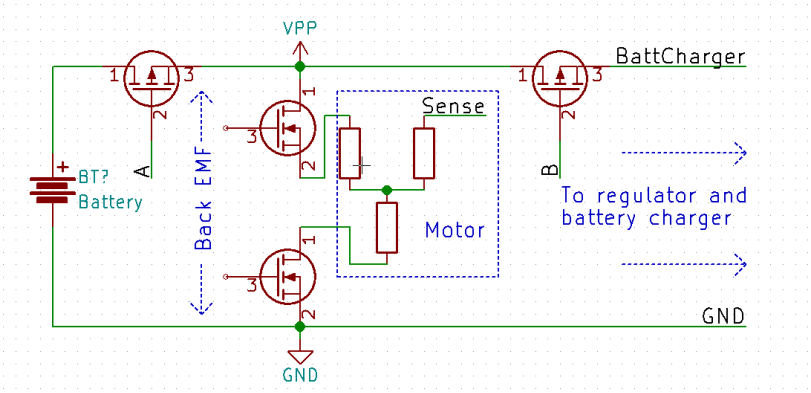

Figure 3-4: By dynamically disconnecting the battery (disabling channel A), the back EMF generated by the propellers, rotating due to water flow, can be directed to a battery charging circuit (enabling channel B).

needing access to a charger. Obviously, this excludes mechanical wear and running out of storage space for data capture.

The final improvement needed to realize unlimited inspection time is a method to provide an external torque to the rotors. This could be achieved through a mechanical "braking" system that affixes the robot to a position along the pipe, with its rotors aligned with the fluid flow, such as the concept shown in Figure 3-5.

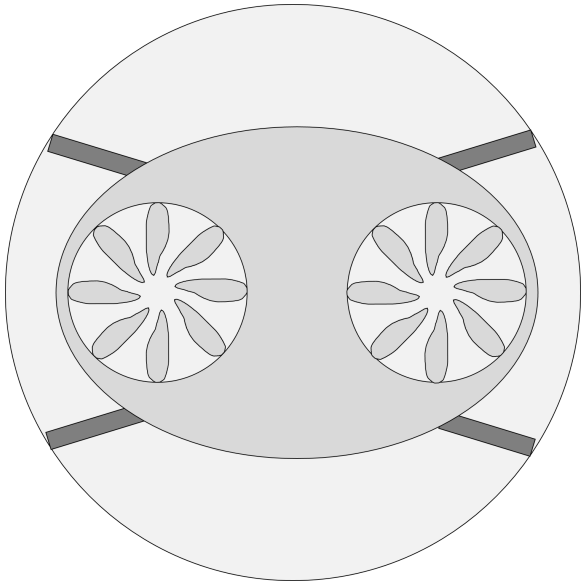

Figure 3-5: The robot could anchor itself to a location in the pipe and use the flow around it as a charging source for unlimited range.

A municipal water supply line experience daily cycles of flow[35], so a robot can

assume there will be fluid flow along most of a municipal water pipeline within an 8-hour interval. Typically there will be fluid flow of some minimal level nearly constantly, so as to reduce bacterial growth and "dead water"[36]. This allows a robot, properly anchored to the pipe wall, to harvest energy from the flow. The rotors are turned by the impinging water, and the regenerative charging circuitry recharges the batteries. This system could be implemented to find areas of high fluid flow when the batteries are running low, so as to minimize charging time and maximize up-time.

3.12 Conclusion

This work developed a full-featured, to-scale controller board to enable the robot to have integrate sensing, computing and actuation. The board was designed to allow the robot to be capable of operating without human intervention, including on-board charging and wireless communication. It includes a 72MHz, 32-bit processor, 4 independent voltage supplies, a USB-compatible LiPo charger, 900MHz radio transceiver, 64Mbit NAND flash memory, two 4A brushless, sensorless motor controllers, and a 9-DOF IMU. A free/libre toolchain was selected to program the robot.

Future work can focus on adding sensing elements to detect pipeline features. This may include a low-resolution camera or short-range proximity sensing to detect pipe walls and junctions. Optical methods are suggested for this, since visible wavelengths travel readily through clean water, and a pipeline environment is typically void of any light sources. The proposed regenerative charging scheme could be implemented. Additional work could also optimize battery characteristics, such as shape and maximum current vs capacity tradeoffs, in order to provide the longest runtime while remaining functional under high current demands from the thrusters.

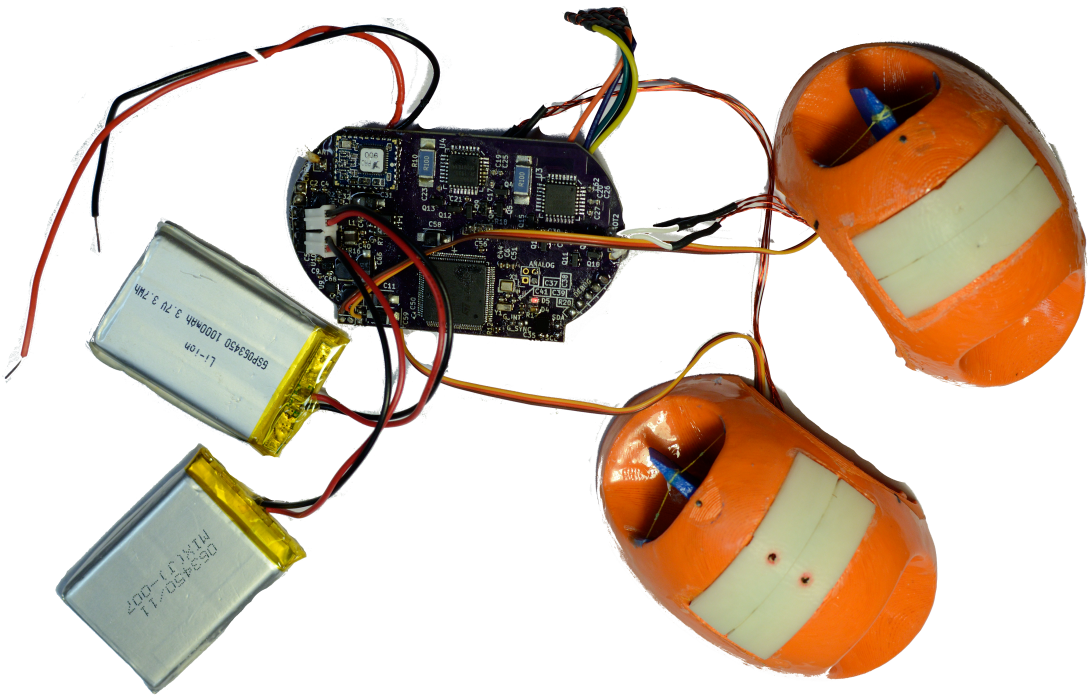

Figure 3-6: Image of the partially assembled electronics. The motors and servos are mounted to the halves of the hull on the right, and the batteries are visible to the left. At center is the custom controller board.

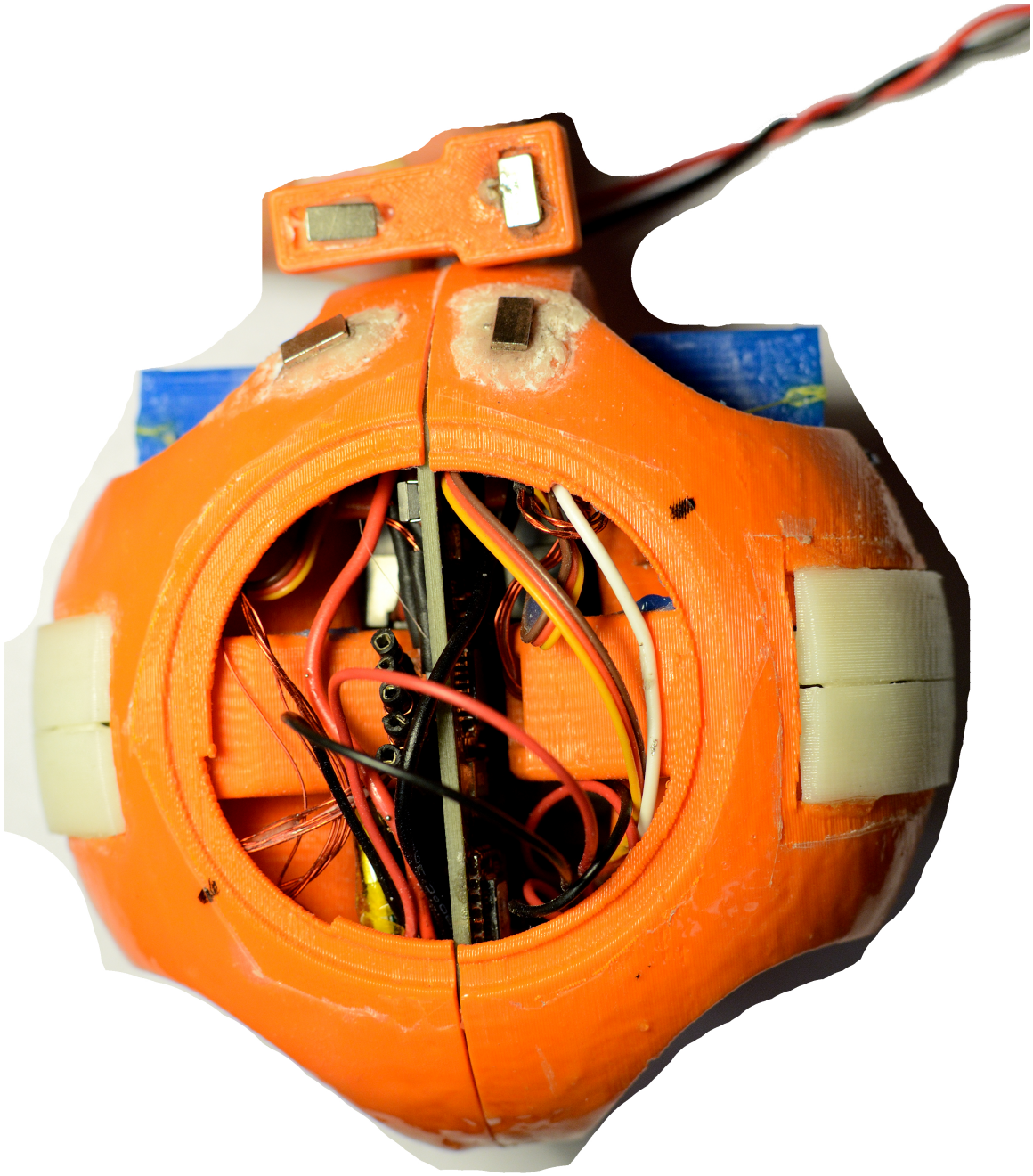

Figure 3-7: Assembled robot, showing how the hull closes with motors, servos, pcb and batteries completely contained. Also shown is the magnetic charger attachment at top, allowing the robot to be charged without unsealing, using a polarized magnetic connector.

Chapter 4

Software Development

The final critical component of the inspection robot platform developed by this work is a comprehensive software system. This system was designed and implemented with future development in mind, and builds a foundation for the qualities needed for a fully autonomous system.

Due to the power and size constraints on the system, the processing power and memory available are both limited, placing more challenging constraints on software development. Given the power available, it was not feasible to implement any sort of GNU/Linux operating system, which would have provided ample access to libraries, as well as a convenient user interface.

Instead a real-time operating system, designed for use on MCUs, was ported to the STM32F446, and middleware and drivers were developed to provide an ecosystem to drive the robot.

4.1 Requirements

Requirements on the software come both as functional requirements driven by desired robot behavior, and as limitations posed by the hardware available. Some limitations guide software development, not through hard, tangible requirements but by informing the kind and style of development that must be employed.

4.1.1 Operating System

Functional requirements include framework and run-time requirements. The software must be capable of multi-tasking, as long-term navigation requires processing at much longer time scales than sensor processing and motor control. This means using an "operating system" (OS) is required, where certain software modules operate the "kernel." The kernel is responsible for selecting which tasks to run and when.

As there are tasks, such as control, that run quickly but must be run regularly, and tasks, such as navigation, that take a comparatively long time to run but may be interrupted, an OS that allows "pre-emption" is required. This allows tasks to be given priorities, and any time a higher priority task is chosen by the kernel to run, its priority is compared to the list of all tasks that currently need running. Out of this list, the task with the highest priority is selected, and any currently running task is halted, allowing the most important task to be running at any given time. This requires care in the creation and assignment of tasks and their priority ratings, as inadvertently giving high-priority to a long-running task can cause the remaining tasks to be stalled longer than is allowable.

This application specifically benefits from a real-time operating system (RTOS) because of the desirability of implementing control systems. Proper implementation of a stable control system requires guarantees on sample spacing in time. An RTOS is specifically designed so that the maximum run-time of any code-path is known, such that tasks of a certain priority have calculable worst-case delay from their scheduled start time. Crucially, this means that no code path can have a while-loop that is not guaranteed to exit. For good system design, peripheral and code faults should be considered in the exit guarantee. Cookbook methods suggest implementing either a timeout for the loop, or a more global watch-dog timer that is capable of resetting failed software components before the maximum allowable delay has elapsed.

4.1.2 Sampling Time Variability

Errors due to small changes in sampling time must be minimized. Failure to maintain a steady sampling frequency results in a lower effective control frequency, degrading performance, and can incorrectly alias input signals if the actual sampling period is not accounted for. In a real-time system, this is readily achievable, but must be designed. The controller task must be the highest-priority task among those with a maximum run-time above some fraction of the control loop period. This will guarantee that the only tasks that may delay or preempt the controller task will be those whose run-times are short enough to be ignored.

4.1.3 Interrupt Service Routines

In conjunction with the above, interrupt routines must also be sufficiently short. The microcontroller ecosystem used has the ability to be "interrupted" during normal program execution by external events, such as changing input pins, completed communications or internal timers. These interrupts are critical components in allowing the software to react quickly to a changing environment, without performing wasteful check-and-wait loops.

In order to react as quickly as possible to external events, the processor will actually stop execution of the current task and jump to a specific "interrupt service routine" (ISR), which runs to completion before the previously running task is resumed. Clearly, ISRs must be designed in such a way that reasonable behavior (which should be enforced through the hardware controlling them) results in a short processing time as well as being called infrequently. More than just the "run-once" time must be considered, since interrupts can be called many times within the time it takes a task to run.

4.1.4 Code Efficiency

Designing a software system around relatively small computing hardware means designing with size and speed in mind. The programmer should be mindful to avoid

premature optimization but the general goals of efficiency should guide the types of structures and practices used.

In the case of an embedded RTOS, particular care should be placed on stack usage. The scheduler is responsible for so-called "context switches," when one task changes to another. A stack, the memory region that stores local variables, function return addresses, and pushed register values, belongs to each task. In the OS used, each tasks' stack space is pre-allocated and a fixed size. The ominous "stack overflow" is when a task uses all of its allocated stack space, and is allowed to access memory that does not "belong" to it. Stack overflow is a real possibility in this environment, with dangerous results to the system if it occurs. Without an always-running kernel intermediary, the software has no way of knowing when a stack overflow has occurred. With the RTOS used, the soonest an overflow can be detected is upon switching tasks. The danger in a stack overflow in this context is the accidental overwrite of a different task's stack. This can easily cause malfunction, or even complete processor lock-up from incorrect function return addresses.

As such, recursive programming is to be avoided, unless a fixed recursion depth can be assigned, since each call adds several words to the stack for saved registers and function return addresses. Functions must return in order to utilize automatic mechanisms for cleaning unused stack. Programmatically pushing to the stack must be done carefully, as this can quickly exceed the available space. Finally, local variables should be kept small; objects like buffers, including strings, should be statically allocated space on the heap, rather than copied to the stack. Frequently used variables should follow the same guideline of static allocation.

Though it was not performed for this work, some RTOS designs in safety-critical systems ensure the prevention of stack overflow through the static allocation of all memory. This allows an automated tool to inspect the compiled code and verify it does not exceed any defined memory limits. However, this obviates dynamic memory allocation, which can be useful in allowing the creation and deletion of large data objects periodically, when concurrent existence would overflow the system.

With RAM limited to 128kB in the current design, yet a relatively large 512kB

of Flash for the program memory, efforts should be made to develop a larger code-base that runs fast, indicating a priority for specialized functionality that improves readability over reuse or overly deep function-calls. To further illustrate the coding style desired, this application is one where a sine look-up table is preferred over an approximation function, using more flash but requiring fewer operations and function calls to get a result.

In particular, future work on navigation will likely push the limits of the processing capabilities of this MCU. It is in the interest of this future work then, that the current work provides a light-weight framework and set of drivers to maximize future processing power.

4.2 Real-Time Operating System

An RTOS was deemed a requirement for the system. It is the only common OS architecture that allows multiple tasks to execute in pseudo-parallel on a single-cored processor, while maintaining guarantees with regard to timing. There are many RTOSs available for use from multiple organizations. They span the space from research projects of unknown stability, to certified, flight-tested software in use by safety-critical OEMs. Implicit in this spectrum is a gradient from free/libre software to tens-of-thousands of dollars for a licensing agreement, respectively. In the interest of costs, present stability, and future portability, a balance of the two extremes was desired.

According to a survey conducted by EE Times, the most commonly used RTOS of 2013 was FreeRTOS[37]. FreeRTOS comes from the SafeRTOS development chain, which is certified by IEC 62304 standard for medical devices, recognized in the EU and the USA. FreeRTOS itself has not been certified to these standards but is developed and supported by the same organization. FreeRTOS is provided without warranty or support, for use in commercial applications, for free. This covers all the aspects desired of an RTOS; low-cost, widely used, and akin to provably-safe design.

Another benefit of selecting FreeRTOS is that it provides support for the STM32F4

processor ecosystem. A reference project was available for one processor in the ecosystem, which aided in porting to the STM32F446. Specifically beneficial for using FreeRTOS with the STM32F4 series is the compatibility for non-managed interrupts. The ARM Cortex-M4 has an integrated interrupt-priority system, allowing for higher-priority interrupts to preempt running, lower-priority interrupts. This benefits the RTOS because a priority-driven system allows some interrupts to have lower priority than the kernel, and some to have higher. The higher-priority interrupts are allowed to interrupt any task at any time, including kernel operations. This provides the lowest latency responses to input. Lower-priority interrupts are allowed to use kernel operations, since the kernel is a higher-priority interrupt. This allows interrupts to use such complex thread-safe structures as queues, inter-process communication (IPC) routines, and task priority management, greatly increasing the utility of a simple interrupt. These features are included in FreeRTOS, and were widely used in the software design.

FreeRTOS is implemented in C, as was the entire project because it is a language that is inherently close to hardware. Allocating variables or memory is a very well-defined process without unknown overhead. The facilities provided by the language itself are deterministic, which is a prerequisite for the software written in it also being deterministic. C allows for nice features such as static analysis and the use of industry-tested tools like GDB for debugging.

The prioritization model of an RTOS informs the software design by suggesting a dichotomy between priority and maximum run-time per characteristic time period. Therefore, any task that needs to run with low-latency or run often must be relatively short, so that it doesn't starve other tasks of processing time. The specific architecture for the software running on top of FreeRTOS is discussed in Section 4.3.

4.3 Architecture

The software architecture is a layered micro-services model, where the behavior and function of specific components is well-defined and compact, visualized in Figure 4-1.

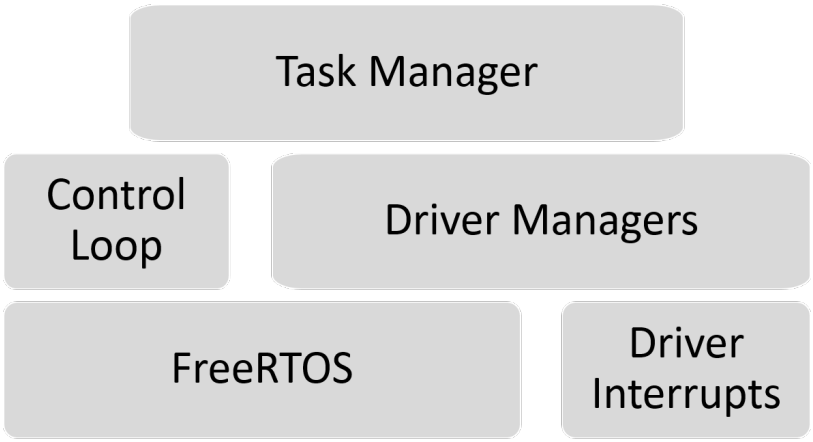

Figure 4-1: The Task Manager is responsible for the most abstract forms of control, and it sends commands to the middle layer managers. These direct the hardware in synchronous tasks, and set up driver interrupts to handle asynchronous tasks.

The base layer is composed of the hardware interfacing services (the "drivers") and the FreeRTOS kernel which provides software services to the higher-level tasks. The middle layer is comprised of the hardware managers, responsible for receiving high-level requests and relaying them to the drivers, and the control system task. The top level is a single task that monitors critical information from each layer below it, handling errors that affect the entire system, controlling startup and shutdown, and servicing high-level requests such as those coming from the radio communications.

One benefit of this model is ease of development, where it is compatible with several key software tools. Primarily, this model allows very simple translation from written software requirement specifications (SRS) to an implemented, functional code-base. An organized list of requirements groups similar functionality together, with sections for specific peripherals or processes. This allows for code and tests to be generated directly from the specification, with all relevant information contained in a section. Larger structures in the SRS can be grouped into files, since C/C++ allows for multiple-file inclusion and compilation.

Compartmentalizing functionality into files is extremely useful to the programmer, as the number of lines of code to be considered at any one time is typically small relative to the size of the project. Additionally, file-based versioning tools can be utilized to track code changes across development cycles. These assist in merging

the work done developing separate areas of the overall software, especially in importing features from one development area that apply to the entire codebase. Finally, tracking changes greatly assists the debugging process, as code can be reverted to a previously functioning state, and the causal changes are documented exactly. Git was the versioning tool used in this project, and much of the success and joy of coding may be attributed to it.

While the model is a structured way of thinking about and developing the software, it also, critically, compartmentalizes functionality into prioritizable segments. As mentioned in Section 4.1.1, an RTOS can give guarantees with regard to timing, but only if requirements are met balancing task priority with run-time. In a microservices model, there's a focus on creating many interconnected tasks in order to contain functionality. Each task can be given its own priority depending on its latency requirements, and in order to fulfill the system timing as a whole. Adding layering to this additionally informs the prioritization of tasks, while allowing for a stratification of run-time as functionality itself gets more abstracted and computationally intense.

4.3.1 Drivers

The base level of the architecture are the drivers that interface with the environment of the system. These drivers generate the outputs requested by higher-level code, and report back the response of the environment. Effectively, they trigger the behaviors designed in all the upper levels. The higher-level code frequently must wait for operations at the driver level to complete; for example, reading values out of the buffer on the IMU. Additionally, it may be critical for the high-level code to formulate a response to some external input quickly, such as if a collision is detected. In both these cases, latency must be at a minimum. This reinforces why the driver-layer is at the bottom of the stack, and as such is given the highest priority.

The drivers are all implemented as interrupts, so as to give near-instant response to stimulus and to use the highest priorities available on the processor. This includes the provided kernel in the FreeRTOS distribution, which uses a timer-based interrupt to check the status of higher-level tasks, and switches the active task if needed.

These drivers are implemented as state-machines, where they spend most of the time inactive. An asynchronous external event can trigger the interrupt through the MCU hardware, such as receiving a byte from the radio. A synchronous request from higher-level code can also trigger the interrupt, typically by placing data in a buffer and enabling a transmit-buffer-empty flag in hardware. The hardware buffer is empty, so the interrupt takes data from the software buffer and loads it into the hardware buffer. As the MCU has several interfaces with complex communication protocols, the state machine requires software to progress through the different aspects of the protocol. An example of this is the IMU communication, where the IMU chip must be transmitted the address from which the MCU would like to read, then communication restarts to read from this address.

A crucial benefit of this design is that complicated communications can be made over relatively slow buses, without the processor having to wait for each byte of the transfer to complete. A high-level task can provide a high-level request, such as store a buffered page to flash, and the interrupt system will handle it entirely, only becoming active when additional interaction is needed. This keeps the driver layer tasks' running time on the order of single microseconds, or a few hundred clock cycles. With a timing demand of accuracy on the order of milliseconds, the activity of the interrupt system can be practically ignored if it is called infrequently per control loop.

This work developed several drivers. The radio link has a driver to automate transmitting and receiving data. The IMU has a driver to handle I2C transactions. The flash memory has a driver to handle reading and writing of specific addresses. The DMA system has a driver to automate sending large quantities of data to the flash driver. The motor controllers have a driver to interpret the automatic reply messages following a command, as well as a driver to convert the motor velocity pulse train into a rotation speed.

4.3.2 Driver Managers and Control Loop

The next layer of the architecture includes the driver manager tasks and the control loop. The driver managers provide the synchronous interface between the rest of the

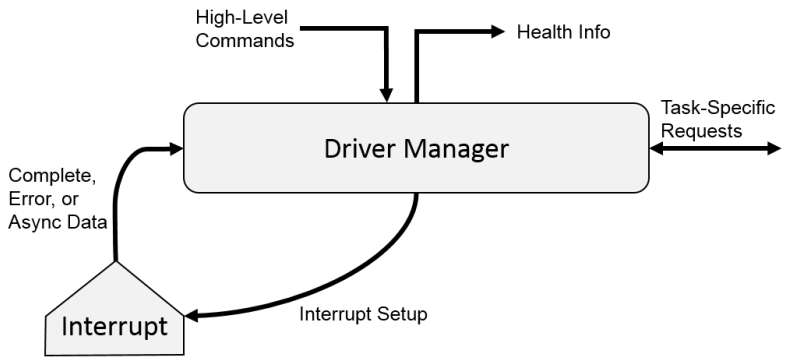

Figure 4-2: The driver managers are a hardware abstraction layer, allowing other tasks to interact with hardware in a thread-safe, non-blocking way without needing to have specific knowledge of the hardware involved. They also allow the high-level code to get health updates for the system.

tasks in the architecture and the asynchronously-activated driver interrupts. Since interrupts are not "called" like functions, they are activated in hardware, the driver manager provides a function-calling interface to pass data to interrupts and activate them. A model of driver manager interaction is shown in Figure 4-2. This abstraction increases simplicity and usability in the higher-level code.

These tasks are generally responsible for tracking the interrupt state so other components of the system can be notified when the interrupt reaches completion. This allows for multiple operations on a single driver to be pending, and handled sequentially and in a thread-safe manner.

The managers also handle communications with protocols that require more processing than the interrupts should manage. An example of this is the flash communication. The flash manager task can control the interrupt and peripheral, or activate the DMA interrupt system and have those two drivers automatically handle the transaction between them. The flash manager can use the calling context, and the current states of the interrupts to determine which interrupts should be activated and how.

Driver managers are also a major component of fault tolerance, responding to errors reported by the interrupts to either retransmit data, reset a peripheral, or simply propagate an error to the top layer so more mission-aware logic can determine a response.

Table 4.1: Task Priorities

Task	Interrupts	Motor	IMU	Controller	Mem	RF	Batt	Master
Priority	4	3	3	2	1	1	1	0

As driver managers are implemented as tasks in the RTOS, they are always running, though can be turned "off," in which case they are inactive and must be woken by another task. Depending on the peripheral, they are designed to do a health-check of the peripheral every 1 to 5 seconds, and otherwise only take action if requested by another task or by their interrupt. This results in them taking very little processing time; they explicitly perform no long-term processing. The task is in an idle state until one of the mentioned inputs awakens it, so most of the time another task can run. These managers have high to medium priority, though lower priority than the interrupts, as their functionality is the basis of all the higher-level code. Sorted priorities for all tasks are shown in Table 4.3.2. Interfacing with peripherals can only occur if the driver managers are given broad preemption capability, otherwise longer-term processing will bottleneck the system.

This layer also holds the control loop, which could be considered the driver manager for the kinematics of the robot. When activated by the top layer, it makes requests to the driver managers for data regarding the robot state, such as rotor rotation rate for thrust calculations and IMU data for body motion, and outputs commands to the motor manager. There is a considerable amount of calculation involved in processing the ideal commands for the given state and desired trajectory, however it is required to have very low latency. Its priority is thus in-between some managers'. For example, the motor controller must be higher priority to guarantee that every command sent by the control loop is executed. Conversely, the flash and data logging manager is one of the longest running managers and lower priority is placed on timely datalogging, so its priority is lower than the control loop task.

This work developed several driver managers. The motor driver manager receives requests for motor power and aileron angle, and reports the motor controller status. The flash manager initializes the flash chip to unlock it for writing, and triggers write sequences or enables the DMA system for automatic writing. The flash manager can

also service requests to read regions of the flash memory. The IMU manager handles initializing the IMU, as well as processing the raw data into pose quaternions. It can also recalibrate the IMU, adjust sensitivity ranges, and generally control the features provided by IMU hardware. The radio manager services transmission requests, and propagates commands received from the base station to the task master. The battery manager runs the battery state machine, monitoring battery voltage and whether a charger is attached, and controls the charging IC and the battery series/parallel arrangement. Lastly, there is the control loop, which reads data from the motor manager and the IMU manager, and generates outputs to the motor manager.

4.3.3 Task Master

At the top of the hierarchy is the task master. This task checks the health of all the drivers through the driver managers, takes high-level commands from the radio manager (such as start test, transmit logged data, restart), and reports the system status back through the radio link for monitoring. These commands are relatively timing-invariant; as long as the system is operating properly, the robot would not be at risk even if "stop test" is processed as much as a second late. As such, the task master, perhaps counter-intuitively, is actually the task with the lowest priority. It performs its function after the timing critical functions have completed, and can be interrupted if any other task is triggered.

The task master also performs the most abstracted form of coordination of the robot's actions. It checks whether the battery driver considers the battery to be in good health, and sends the shutdown command to all drivers if it isn't. This allows for system-level logic to be performed, where the state of all tasks can be considered before taking action. Though this work only touched the surface of failure handling for this robot, having system-level logic can greatly improve the fault-detection and fault-tolerance of a system. For example, a top-level taszk is required to handle a fault like loss of a thruster. The robot could be put into an "emergency mode" of operation, with the control loop notified that a thruster is disabled. Depending on the battery state-of-charge, the current limit on the lost thruster could be disabled to try

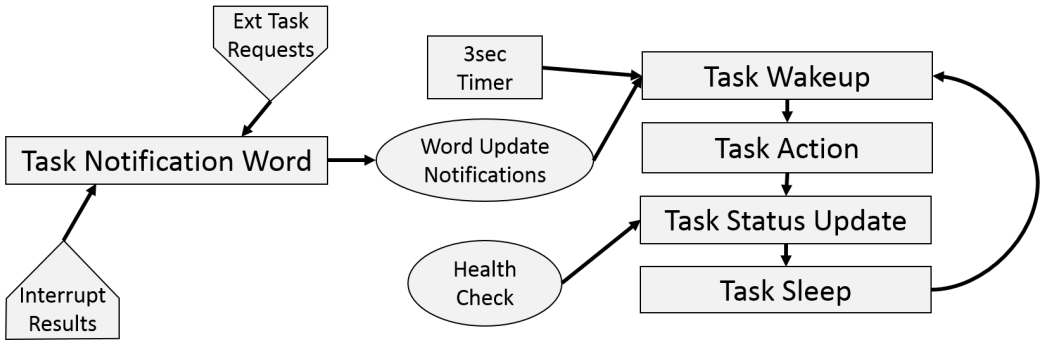

Figure 4-3: FreeRTOS provides an IPC mechanism that allows interrupts and other tasks to post notifications to another task. The task can sleep until either it receives a notification, or a timeout occurs. In the task model developed, tasks sleep for a few seconds, waiting for notifications. Notifications cause actions to occur, such as enabling an interrupt. Notification or not, a health check is performed and the task status updated before going back to sleep.

and force it to move, or a more power-saving control action could be communicated to the remaining thruster. Since this logic bridges multiple drivers, it is only appropriate to build it externally.

4.4 Task Model

Since several task managers were implemented, a unified framework was developed to provide matching interfaces and behaviors between them, diagrammed in Figure 4-3 and with example code in Appendix A. This greatly eased development time for each task, and provided a set of well-tested idioms that could be used confidently.

The first idiom is the manager status. Each manager has a state machine that depends on the current status and some set of inputs to determine the manager's next status. The array of possible statuses varies according to the possible states of the manager but 3 key statuses are always present; OFF, ON and ERROR. Additionally, each manager has both an internal status and an external status. Since external tasks don't need the verbosity of all the possible states in the manager's internal state machine, one or more internal statuses map to a single external status. Typically this external status will be one of the three guaranteed statuses; OFF, ON or ERROR.

Some managers implement a BUSY status externally to show that further input to the manager should be delayed until the next status check; this occurs in the radio manager if the outgoing data buffer is full. Internally, managers are free to have as many statuses as required for meaningful state transitions.

To utilize the internal status, all managers implement a status transition process. When the manager is not being called internally or externally by drivers or tasks, it reawakens periodically to perform a health check. Based on the results of the health check, the manager can change its status to ERROR, or remain in the status it was in. The status change function prevents ERROR statuses from being cleared; to do this a specific function with permission to clear errors must be called. This allows the ERROR state to persist (and be properly handled by downstream logic in the manager) until a specific and manager-dependent error-clear function has been executed. This may include resetting the task state, or even completely restarting the peripheral it is managing. The same status transition process is what allows the manager to handle requests from drivers or tasks as soon as they are received and the manager becomes the highest priority task.

FreeRTOS provides a task notification feature that incorporates very useful IPC functionality. Essentially, external tasks can modify a task's notification word to give it information. That task can sleep until either a notification is received or a timeout elapses. All managers implement this; the timeout causes a standard health check, while a notification produces some action. With 32-bit notification words, there is enough room for each notification a manager might receive to be indicated with a specific bit in the word. This allows several requests to be handled even if they are received simultaneously, which does occur during interrupt behavior. Examples of requests are shutdown and startup requests from the task master during battery charging, and peripheral unlocking requests once a driver reports an operation has been completed.

4.5 Conclusion

This work developed a software ecosystem and extensible framework for the control of an autonomous pipeline inspection robot. An RTOS was selected as the basis for a threaded operating system running on the single-core processor. Drivers were developed to control the various hardware peripherals integrated into the system. A generic framework for hardware and abstraction managers was developed to create common interfaces, software idioms, as well as an RTOS-synergetic program flow. A simple control-loop was implemented as a stand-in for further development and testing on the platform. A task master was developed to manage high-level commands from a human-interfacing base-station, and to provide high-level autonomy and safety direction for the managers.

Future work is suggested to explore more detailed interpretation of the IMU data, including integrating a Kalman filter to better reject noisy or drifting measurements. The robot's motion should be analyzed to determine its behavior in fluid flow, and a controller developed to direct its motion. High-level navigation software for pipelines has been developed to analyze a pipe network as a graph, which simplifies long-range navigation as simply a graph-traversal problem[38]. Additional work could be done to convert this to a compact and efficient form on-board this robot. As many cities lack complete or current maps of their pipelines, future work could implement Simultaneous Localization and Mapping (SLAM) on the robot, such that it could investigate and update the cities maps while inspecting.

Chapter 5

Conclusions and Recommendations

This work covers the incremental improvement of a pipeline inspection robot platform under development at the Mechatronics Research Laboratory at MIT. Three facets of the design have been improved or originally developed.

5.1 Summary

Section 2 covers the development of an out-of-plane actuation system to give the robot controllability for vertical translation, roll and pitch. Thrust vectoring, additional thrusters, gyroscopic actuators and attitude-adjustment nozzles were considered but deemed too complex or space inefficient. Instead, a design for ailerons that impinge on the flow from the main thrusters was created and implemented on the robot.

Section 3 describes the development and design of a dedicated, optimized embedded controller board to fit inside the robot housing. Previous work used space-inefficient off-the-shelf parts, which lacked the complexity and robustness required by an autonomous robotics platform. A comprehensive mobility platform was designed, including an IMU, motor controllers, data-logging memory, wireless communications and a 72MHz 32-bit microcontroller. The design was fabricated in hardware, and successfully integrated with the physical system.

Section 4 details the design and implementation of a general software framework for command and control of the robot and its associated hardware. An architecture was

chosen to simplify and unify the design of specific software elements. A task structure unifies interfaces between software components and provides well-tested and stable IPC and program flow. The developed firmware is capable of receiving high-level commands from a human-interface terminal to drive the robot or download captured data.

In summary, this work develops the robot to be capable of movement in a general aquatic environment, with integrated sensing and control, in order to perform mobility and control system testing, with room for future autonomous navigation capability.

5.2 Recommendations for Future Work

With an integrated development platform ready, future work should focus on navigation and control of the robot. Additional sensing may be needed to identify pipe walls and pipeline features such as junctions. The hydrodynamic behavior of the robot needs to be verified in different fluid regimes; from static surroundings to restricted flows inside pipes. Additional testing may be needed to evaluate the long term water-tightness of the aileron control line design. The concept of charging the robot by anchoring and using the surrounding flow would be very useful in increasing its range. Development of a space- and power-efficient anchoring system is needed.

Appendix A

Model Task Code

```
#include "task.h"

typedef enum {
  TASK_INT_READY, //task ready
  TASK_INT_BUSY,  //task being set up, or errors being clear/handled
  TASK_INT_OFF,   //task turned off, generally to await power ok
  TASK_INT_ERROR  //unrecoverable error, reset needed
} task_state_internal_t;

//task is off until successful initialization
static task_state_internal_t state = TASK_INT_OFF;

//This is the main thread for the task
void task(void * pvParams)
{
  //perform any startup initialization
  task_setup();

  //allocate space for the task notification word
  uint32_t note;

  while(1){

    //tasks run forever, sleeping if they have nothing to do
    //check our notifications, or timeout if there are none within 2 seconds

    if(xTaskNotifyWait(0, TASK_REQ_MASK, &note, 2) == pdTRUE){
      //do some state transitions
      if(!(note & TASK_REQ_MASK)){
        //handle invalid notification
        task_set_state(&state, TASK_INT_ERROR);
        continue;
      }

      if(note & TASK_REQ_UPDATE){
        //the task was asked to wake up, continue
        ;
      }

      if(note & TASK_REQ_PERIPHERAL_ISR){
```

```
        //the ISR woke the task up
        do_something();
      }

      if(note & TASK_REQ_CLEAR_ERR){
        //task master wants to reset to clear error
        if(state == TASK_ERROR){
          task_reset();
        }
      }
      if(note & TASK_REQ_ON){
        //restart after shutdown
        if(state == TASK_OFF){
          task_setup();
        }
      }
      if(note & TASK_REQ_OFF){
        //shutdown task, low power mode
        if(state == TASK_READY  || state == TASK_BUSY){
          task_shutdown();
        }
      }
    }

    if(state == TASK_READY){
      do_stuff();
    }

    //do a status update while we are awake
    if(task_get_diag()){
      task_set_state(&state, TASK_INT_BUSY);
    }
  }
}

static void task_set_state(task_state_internal_t new_state)
{
  if(state == TASK_INT_READY || state == TASK_INT_BUSY){
    //some transitions can happen any time
    state = new_state;
  } else if(state == TASK_INT_OFF){
    //some transitions are restricted
    if(new_state != TASK_INT_BUSY){
      state = new_state;
    }
  }
  //no case for error, errors can only be cleared by calling clear error
}

static void task_clear_error(void)
{
  if(state == TASK_INT_ERROR){
    state = TASK_INT_OFF;
  }
}

task_state_t task_get_state(void)
{
```

```
  //allow external states to vary from internal states
  switch(state){
    case TASK_INT_READY:
      return TASK_READY;
    case TASK_INT_BUSY:
      return TASK_BUSY;
    case TASK_INT_OFF:
      return TASK_OFF;
    case TASK_INT_ERROR:
      return TASK_ERROR;
    default:
      return TASK_ERROR;
  }
}

void task_router(uint8_t type, uint8_t *data_buff)
{
  //allow the radio communications task to deliver payloads
  //into this task for handling
  switch(type){
    //type byte can indicate a class of message
    case 0:
      //values or other data can be passed in using the data_buff array
      do_something(*data_buff)
      break;
    case 1:
      do_something_else(*data_buff);
      break;
    default:
      //unknown message type
      task_set_state(&state, TASK_INT_ERROR);
  }

  //since this function is called by another task, this task
  //needs to be woken up to handle the new data
  xTaskNotify(task_handle, TASK_REQ_UPDATE, eSetBits);
}

void task_setup(void)
{
  //perform startup initialization
  do_startup();

  task_set_state(&state, TASK_INT_READY);
}

void peripheral_isr(void)
{
  //set a variable to store whether kernel ops would awaken a higher
  //priority task, assume no to start
  BaseType_t lHigherPriorityTaskWoken = pdFALSE;

  //read the status word from the peripheral, eg the QUADSPI
  uint32_t status = QUADSPI_SR;

  //if it's some value, do something
  if(status & QUADSPI_SR_TEF){
    do_something();
  }
```

```
    //wake the task up so it can deal with new info
    //tell it that this interrupt is the reason it woke up
    //check if doing this would make a higher priority task
    //wake up, meaning we DO NOT jump back to the same task
    //when the ISR is done
    xTaskNotifyFromISR(task_handle, TASK_REQ_PERIPHERAL_ISR,
            eNoAction, &lHigherPriorityTaskWoken);

    //go to whatever the high priority task is
    portEND_SWITCHING_ISR(lHigherPriorityTaskWoken);
}
```

Appendix B

Circuit Schematics

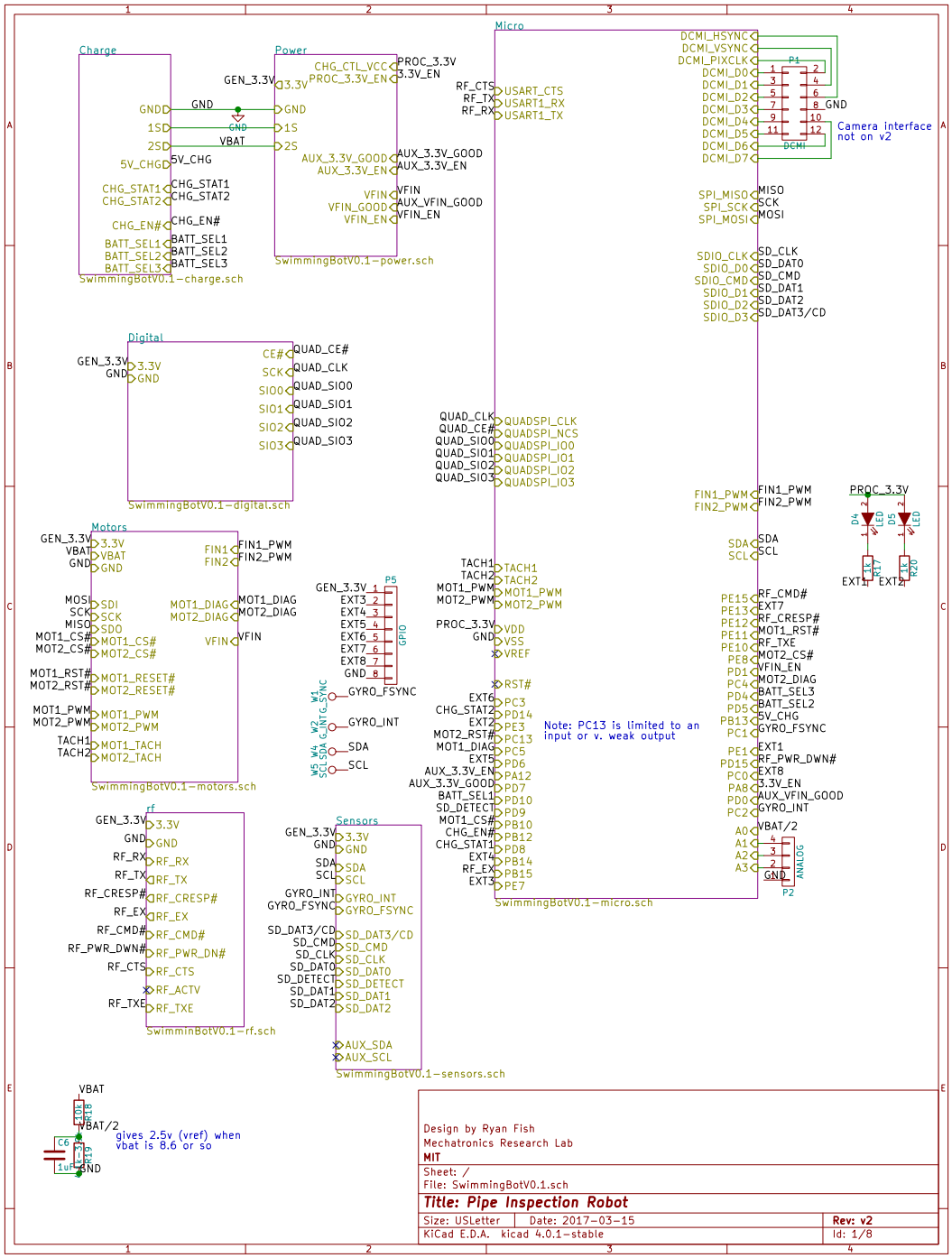

Figure B-1: Subsystem Connection Schematic

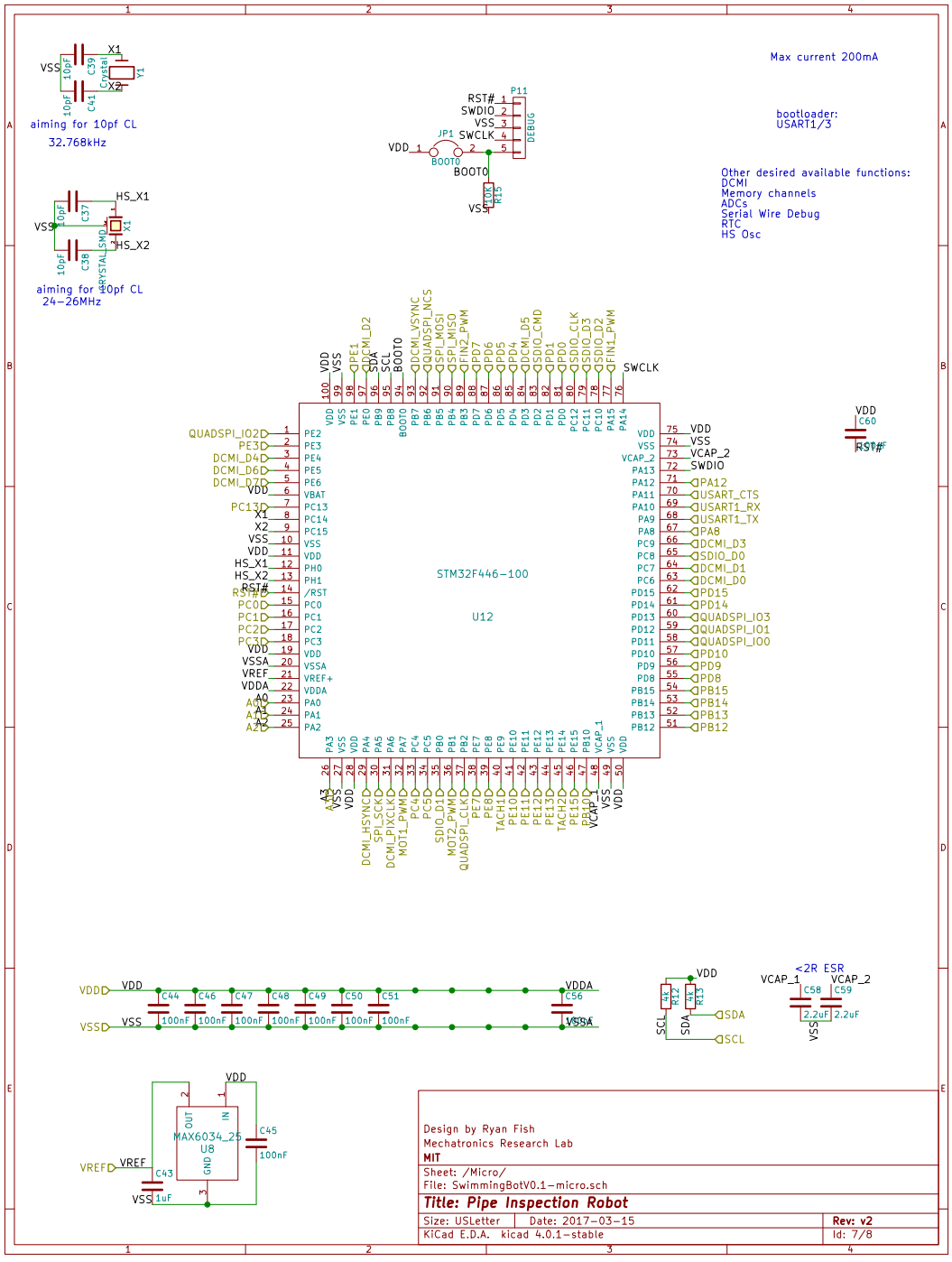

Figure B-2: Microcontroller Subsystem Schematic

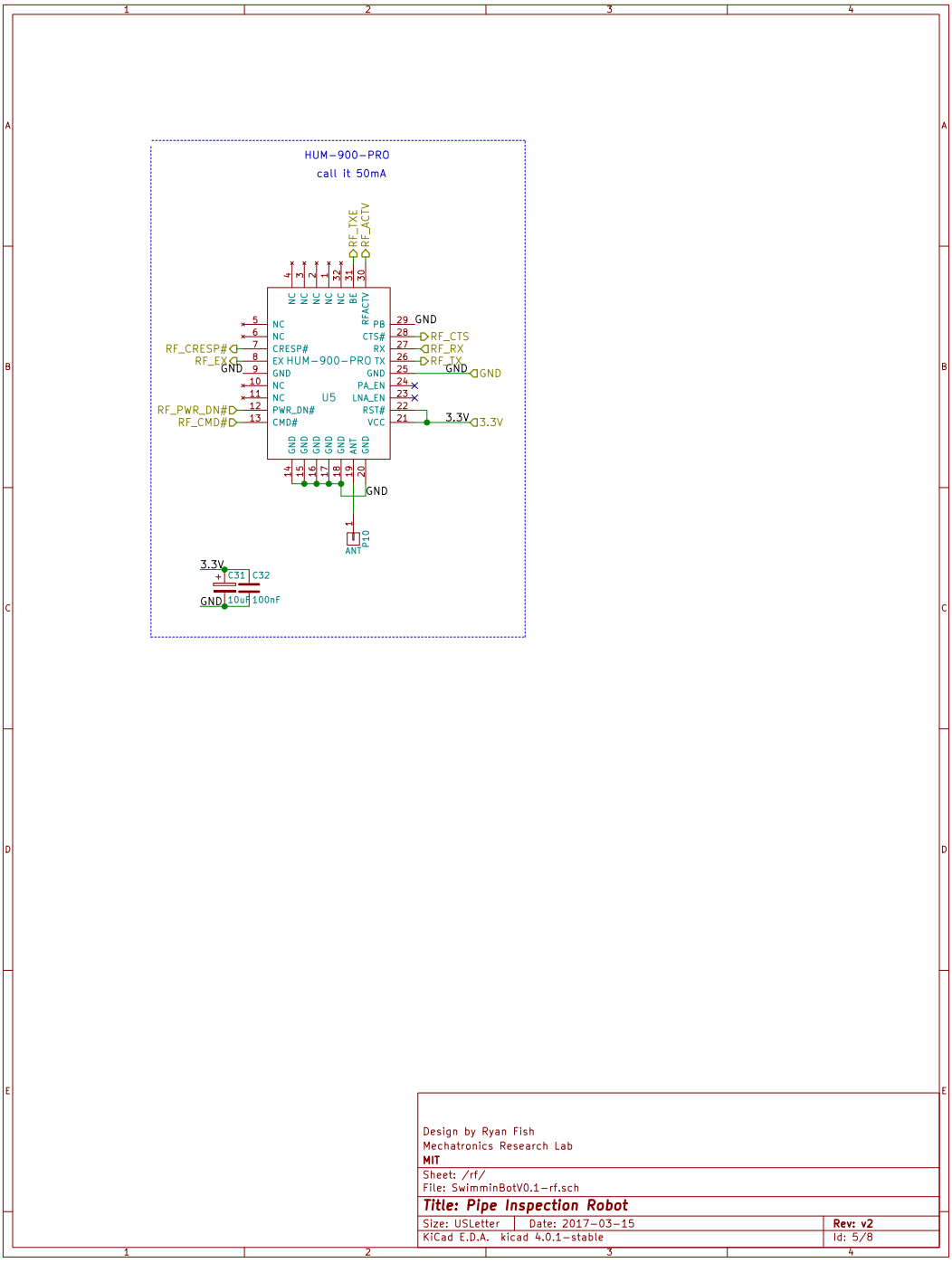

Figure B-3: Radio Communication Subsystem Schematic

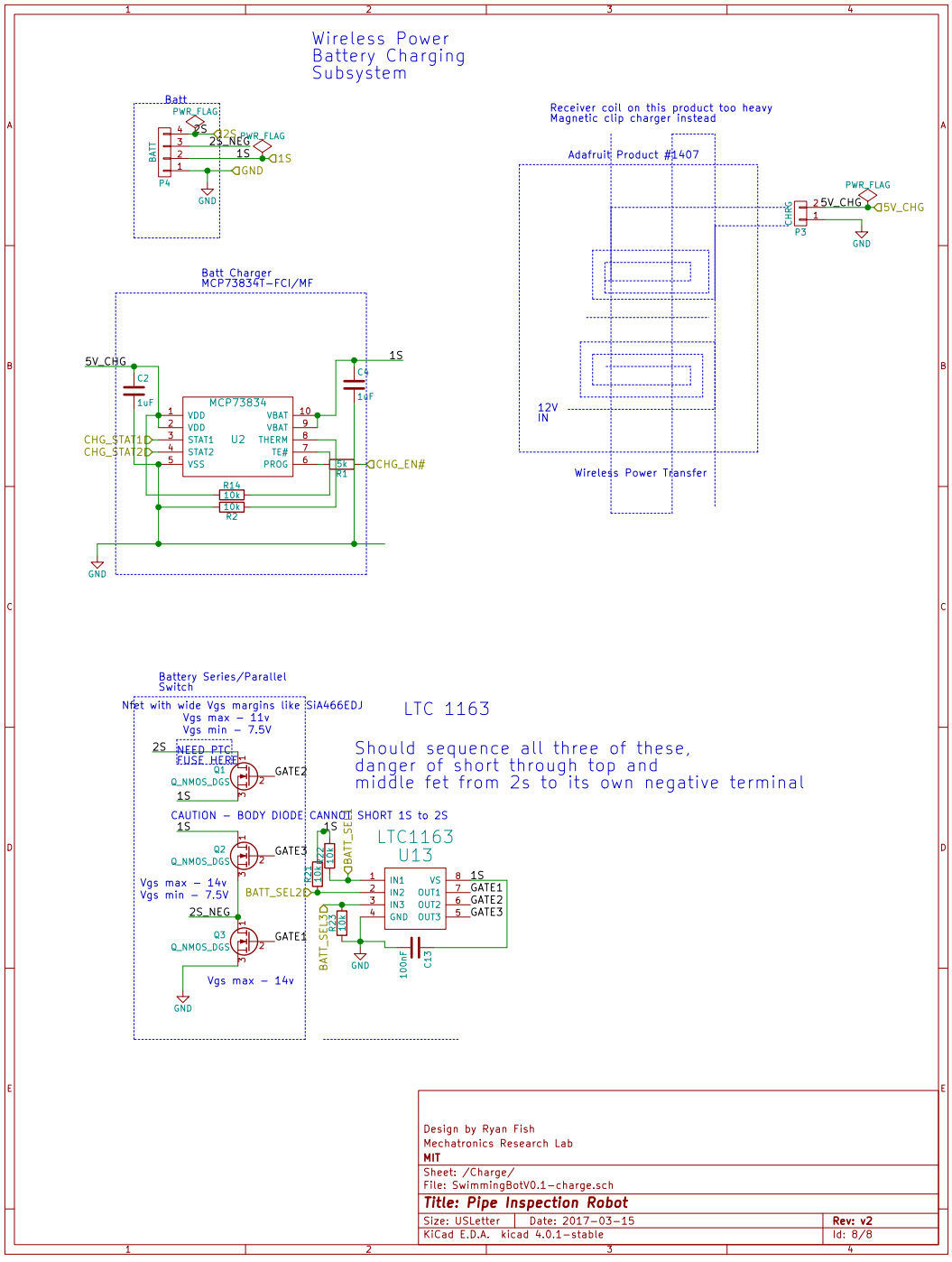

Figure B-4: Charging and Battery Conversion Subsystem Schematic

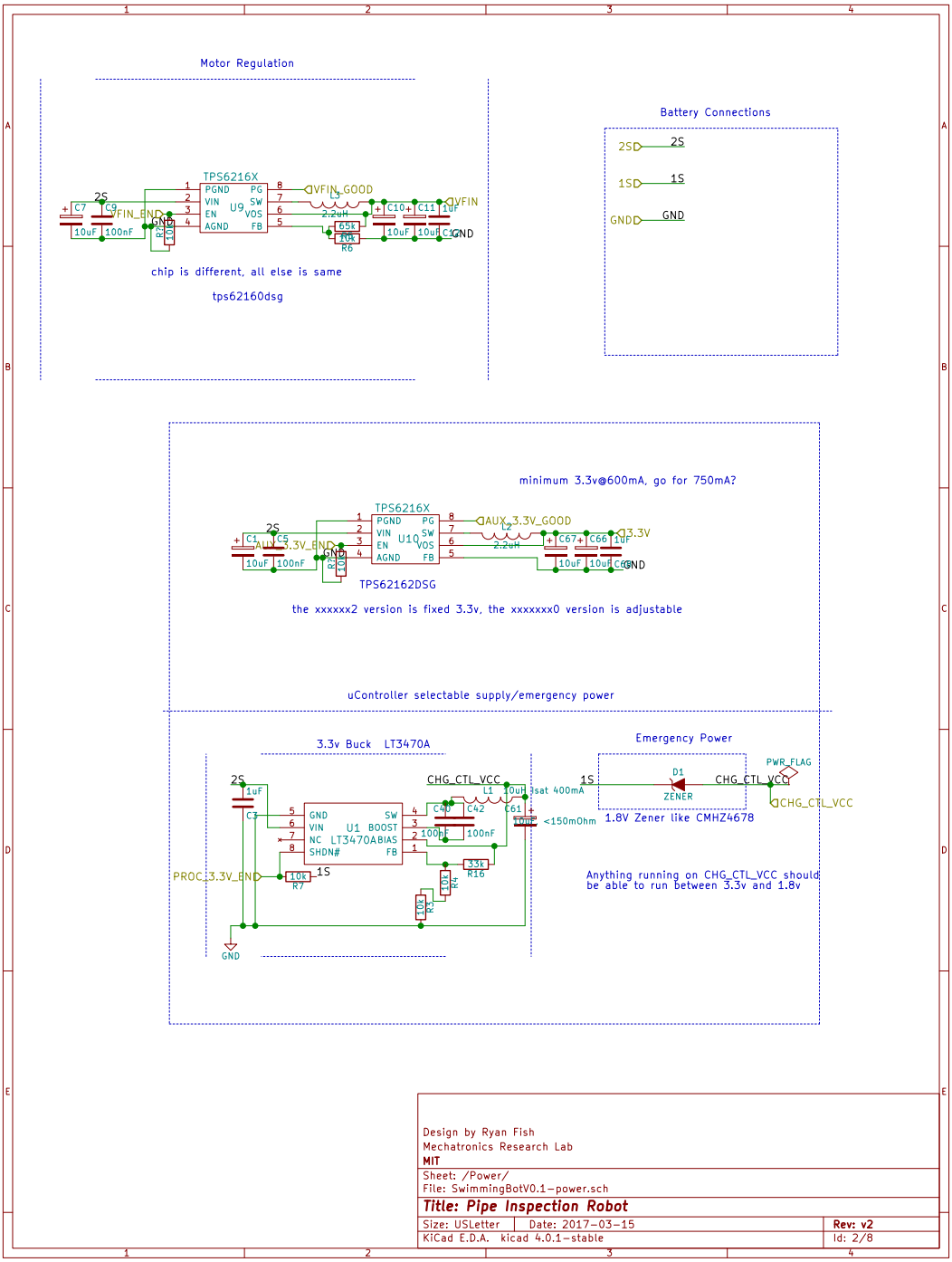

Figure B-5: Power Subsystem Schematic

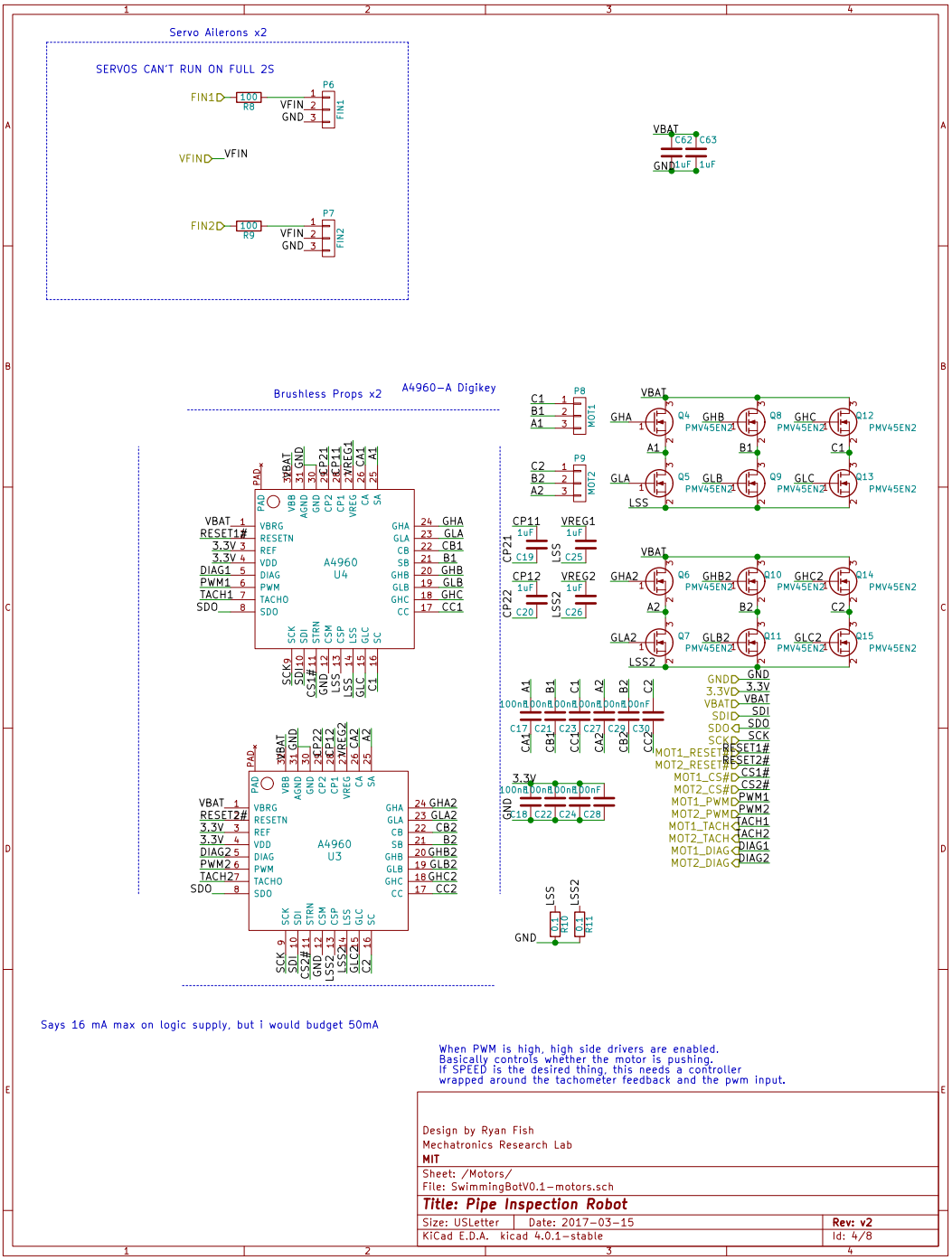

Figure B-6: Motor Driver Subsystem Schematic

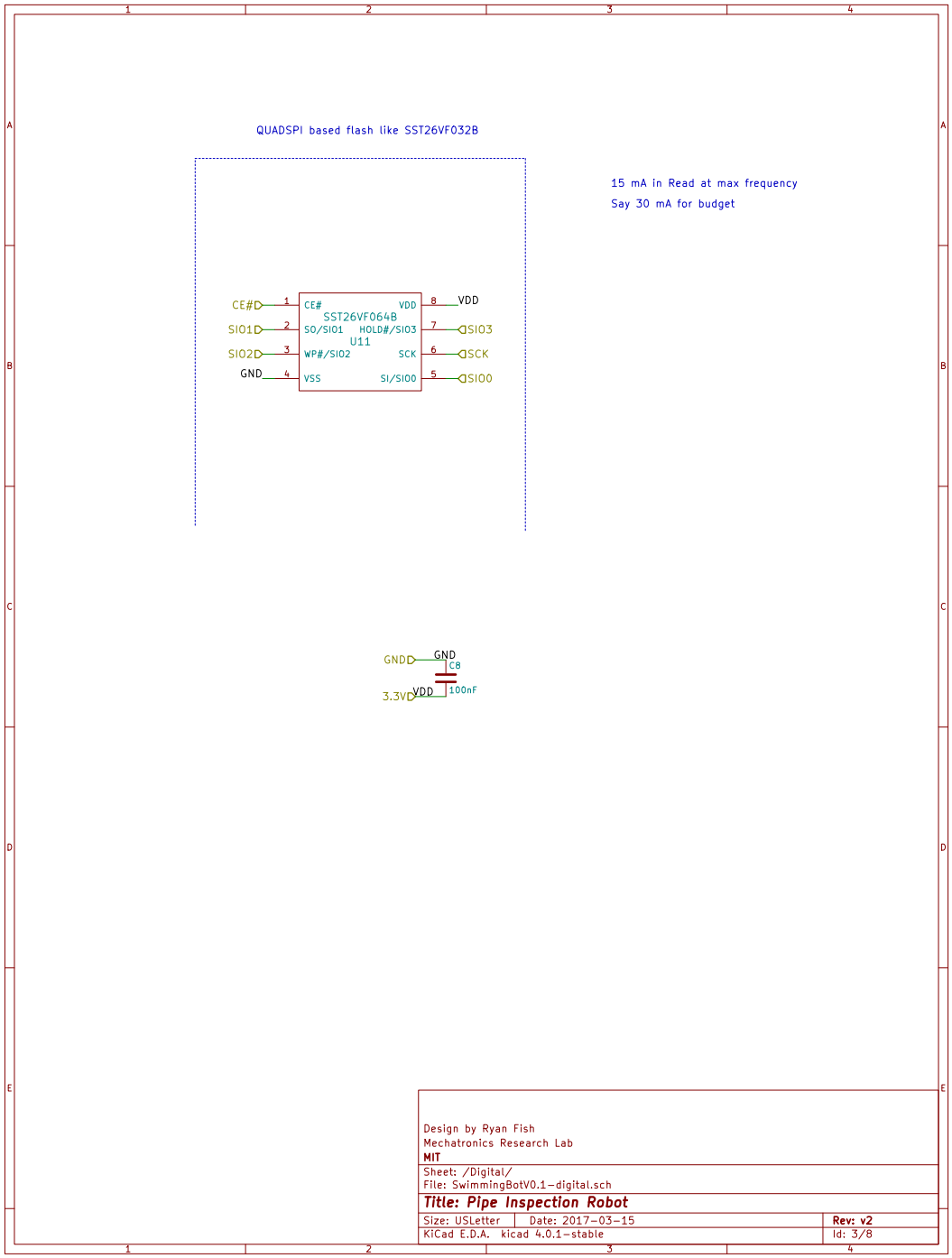

Figure B-7: Memory Subsystem Schematic

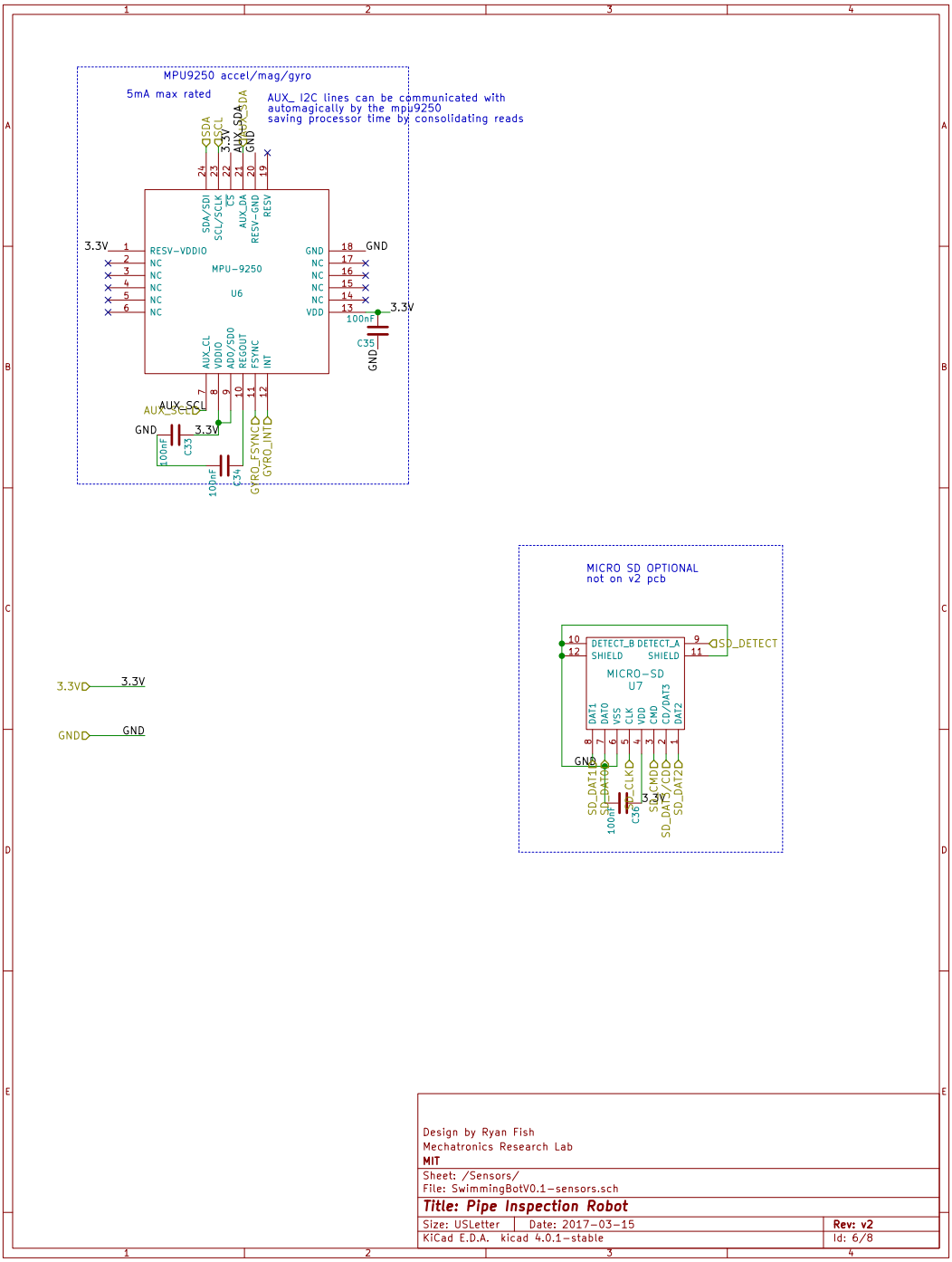

Figure B-8: Sensor Subsystem Schematic

Bibliography

[1] N. Snow, "ELI forum examines pipeline, rail, maritime crude oil transportation," *OIL & GAS JOURNAL*, vol. 112, pp. 25–28, May 2014.

[2] Federal Energy Regulatory Commission, "FERC: Industries - General Information," Apr. 2004. https://www.ferc.gov/industries/oil/gen-info.asp

[3] S. Foster, "Urban water-supply security - making the best use of groundwater to meet the demands of expanding population under climate change," *Groundwater and climate in Africa. Proceedings of the Kampala Conference, Uganda*, June 2008.

[4] *Drinking water infrastructure needs survey and assessment: fifth report to Congress.* Washington, DC: U.S. Environmental Protection Agency, Office of Water, Office of Ground Water and Drinking Water, Drinking Water Protection Division, 2013. https://www.epa.gov/sites/production/files/2015-07/documents/epa816r13006.pdf

[5] "Infrastructure Report Card 2017: Drinking Water," Jan. 2017. http://www.infrastructurereportcard.org/wp-content/uploads/2017/01/Drinking-Water-Final.pdf

[6] K. R. Piratla, S. R. Yerri, S. Yazdekhasti, J. Cho, D. Koo, and J. C. Matthews, "Empirical Analysis of Water-Main Failure Consequences," *Procedia Engineering*, vol. 118, pp. 727–734, Jan. 2015.

[7] Jordiferrer, "Damage caused by a pipe burst at carrer Ganduxer, Barcelona. July 2016," July 2016. https://commons.wikimedia.org/wiki/File:2016_Canonada_d%27aigua_rebentada_al_carrer_Ganduxer_01.jpg

[8] "United States of America, Code of Federal Regulations, 49 CFR 192.i."

[9] Audriusa, "A station on a natural gas pipeline used for the loading and unloading of pipeline inspection gauges, commonly known as "pigs". Photographed near Dubendorf, Switzerland.." https://commons.wikimedia.org/wiki/File:Pipeline_device.jpg

[10] H. Barrison, "Used to clean the pipeline. This one is displayed for tourists.," Aug. 2007. https://commons.wikimedia.org/wiki/File:Trans-Alaskan_Pipeline_pig.jpg

[11] J. Chin and E. Fakas, "Evaluations of Surface And Subsea Pig Launching Systems," in *ISOPE-I-04-177*, (ISOPE), International Society of Offshore and Polar Engineers, Jan. 2004.

[12] C. Armstrong, *Rehabilitation of water mains*. Manual of Water Supply Practices: M28, Denver, CO: American Water Works Association, 2014.

[13] *Distribution system requirements for fire protection*. AWWA manual: M31, Denver, CO: American Water Works Association, 2008.

[14] Y. Wu, "Design and fabrication of a maneuverable robot for in-pipe leak detection," Master's thesis, MIT, Cambridge, MA, 2014.

[15] G. P. Sutton and O. Biblarz, *Rocket propulsion elements*. Hoboken, NJ: Wiley, 2010.

[16] S. Wilkinson, "Jump Jet," *Aviation History*, vol. 26, pp. 20–29, Mar. 2016.

[17] R. M. Gerdes, "Lift-fan aircraft: Lessons learned-the pilot's perspective," tech. rep., NASA Ames Research Center, Aug. 1993. `https://ntrs.nasa.gov/search.jsp?R=19940011477`

[18] W. E. Graf, "Effects of Duct Lip Shaping and Various Control Devices on the Hover and Forward Flight Performance of Ducted Fan UAVs," Master's thesis, Virginia Tech, Blacksburg, VA, June 2005. `http://theses.lib.vt.edu/theses/available/etd-05262005-170916/`

[19] K. M. Khan, *Fluid mechanics and machinery*. New Delhi: Oxford University Press, 2015.

[20] K. J. Butcher, *Reference data : CIBSE guide C*. Oxford: Butterworth-Heinemann, 2007.

[21] P. M. Sforza, "Appendix B: 1976 US Standard Atmosphere Model," *Commercial Airplane Design Principles*, pp. 487–495, Jan. 2014.

[22] John W. R. Taylor, *Jane's All the World's Aircraft, 1965-1966*. Sampson, Low, Marston & Co Ltd., 1965.

[23] L. F. Belew, *Skylab, our first space station*. NASA SP: 400, Washington, DC: Scientific and Technical Information Office, National Aeronautics and Space Administration, 1977.

[24] Cleonis, "Kreisel in kardanischer Aufhängung. Beschriftet für Erklärung der Kreiselmomente als direkte Folge von Coriolis-Kräften. Kann Präzession und Selbstfindung der Nordrichtung von Kreiselkompassen verdeutlichen.," Apr. 2007. `https://commons.wikimedia.org/wiki/File:Gyroscopic_precession_256x256.png`

[25] D. A. Fester, "Space shuttle reaction control subsystem propellant acquisition technology," *Journal of Spacecraft & Rockets*, vol. 12, pp. 705–710, Dec. 1975.

[26] NASA, "Apollo LM (Lunar Module) RCS (Reaction Control System) quads," Dec. 1972. https://commons.wikimedia.org/wiki/File:LM_RCS.jpg

[27] S. Bhattacharyya and H. H. Asada, "Control of a compact, tetherless ROV for in-contact inspection of complex underwater structures," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, p. 2265, 2014.

[28] M. F. Schiavo, "History of human flight," *Salem Press Encyclopedia*, 2016.

[29] P. S. Foresman, "line art drawing of an aileron on a wing," Aug. 2007. https://commons.wikimedia.org/wiki/File:Aileron_(PSF).jpg

[30] J. Miller, *The X-planes, X-1 to X-29*. Marine on St. Croix, MN: Specialty Press, 1983.

[31] L. J. Clancy, *Aerodynamics*. United States: Wiley, 1975.

[32] D. Ma, Y. Zhao, Y. Qiao, and G. Li, "Effects of relative thickness on aerodynamic characteristics of airfoil at a low Reynolds number," *Chinese Journal of Aeronautics*, vol. 28, pp. 1003–1015, Aug. 2015.

[33] "Diagram showing the linkage between the control column and wheel and the various control surfaces of the Pfitzner Flyer," Mar. 1910. https://commons.wikimedia.org/wiki/File:PfitznerFlyer_Control_linkage_FlightGlobalVoll_II(9)_p.182.jpg

[34] Honeywell International, Inc., "Spectra Fishing." http://www.spectrafishing.com/

[35] B. J. Carragher, R. A. Stewart, and C. D. Beal, "Quantifying the influence of residential water appliance efficiency on average day diurnal demand patterns at an end use level: A precursor to optimised water service infrastructure planning," *Resources, Conservation & Recycling*, vol. 62, pp. 81–90, May 2012.

[36] Jeffrey Denoncourt, "Best Practices for Water Purification System Design," *Plumbing Systems & Design*, pp. 28–35, Jan. 2007.

[37] Peter Clarke, "Android, FreeRTOS top EE Times' 2013 embedded survey," Feb. 2013. http://www.eetimes.com/document.asp?doc_id=1263083

[38] K. Jha, "Minimal loop extraction for leak detection in water pipe network," in *2012 1st International Conference on Recent Advances in Information Technology (RAIT)*, pp. 687–693, Mar. 2012.