Design of a Deformable Spherical Robot

by

Nathaniel Justin Lee

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

February 2022

Signature of Author: _____

Department of Mechanical Engineering
February 4, 2022

Certified by: _____

David E. Hardt
Ralph E. and Evelyn F. Cross Professor of Mechanical Engineering
Thesis Supervisor

Accepted by: _____

Kenneth Kamrin
Associate Professor of Mechanical Engineering
Undergraduate Officer

Design of a Deformable Spherical Robot

by

Nathaniel Lee

Submitted to the Department of Mechanical Engineering
on February 4, 2022 in Partial Fulfillment of the
Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

ABSTRACT

The design of a uniquely actuated point-supported spherical robot is investigated to support the claim that deformable spherical robots that are point-supported can increase the speed of the robot. The spherical robot design investigated in this thesis has a unique actuation system because it uses a low number of actuators with the goal of increasing average speed and reducing weight. The most relevant design requirements for this robot were to follow a 40cm/s trajectory and weigh 10 kg or less. The overall design to meet these design requirements and others is discussed and is further explained how the proposed robot functions.

A dynamic model was developed to determine a control system for the robot, and to verify the design. The robot was modeled in Simulink using this control system to determine if the 40 cm/s was feasible and to determine if the selected actuators meet the simulated torque and speed requirements. The torque and speed requirements were higher than anticipated for some of the selected actuators in the design presented in this work. When point-supported, the robot was able to follow 40 cm/s circular trajectory in a Simulink model. The robot was also modeled without the parts that make it point-supported, so that a comparison could be made between the two configurations. The robot in the configuration that was not point-supported followed the trajectory much more slowly, supporting the claim that point-supported deformable spherical robots can be faster.

Thesis Supervisor: David E. Hardt
Tile: Ralph E. and Evelyn F. Cross Professor of Mechanical Engineering

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

Spherical robotics is a branch of robotics that takes advantage of spherical geometry to navigate in an environment. The spherical geometry allows for movement in any direction regardless of orientation, unlike a differential drive robot, for example, which must change its orientation to move in a different direction.

There are several ways to actuate a spherical robot. In (Chase et al. 2012) three main "principles used to propel a spherical robot" were reviewed: barycenter offset, conservation of angular momentum, and shell transformation. Barycenter offset spherical robots move by shifting their center of mass inside of a spherical shell resulting in the robot rolling in the desired direction. Barycenter offset designs are the easiest to implement because of their simplicity in design and control. The conservation of angular momentum principle uses control moment gyroscopes to actuate the robot by taking advantage of the law of conservation of angular momentum.

In shell transformation (the focus of this thesis) the spherical robot deforms its outer shell to effectively roll (Chase et al. 2012). In (Nozaki et al. 2019) three main types of shell transformation or shape changing spherical robots are defined: radial skeleton, edge skeleton, and tensegrity. An edge skeleton spherical robot has several prismatic joints on the edges of a polyhedron to deform and roll. Tensegrity robots use a combination of rods and cables to change the positions of the polyhedron vertices that its geometry is modeled after. Lastly, radial skeleton spherical robots use prismatic joints that point radially outwards from a polyhedron center (Nozaki et al. 2019). The radial skeleton type of shell transformation spherical robot will be investigated here.

There are two types of rolling forms in shape changing spherical robots: continuous and discrete. Continuous or "free form locomotion", proposed by (Nozaki et al. 2019), allows for a smoother trajectory that does not rely on the contact area geometry. Discrete rolling follows a trajectory that does depend on the contact area geometry of the overall polyhedron that the robot is modeled after (Nozaki et al. 2019). Discrete rolling propels a spherical robot in a set number of directions in steps instead of continuous rolling that can be achieved by barycenter offset and conservation of angular momentum designs (Chase et al. 2012) (Nozaki et al. 2019).

The robot considered in this work is a discrete rolling radial skeleton spherical robot. This robot has a unique geometry because it uses rounded bumpers to form a sphere so that the robot is effectively point-supported. The point-supported aspect is important because it allows the robot to move faster and follow a smoother trajectory than a polyhedron face supported robot. This work also presents a novel method for actuating a point-supported discrete rolling radial skeleton spherical robot.

1.1 Motivation

The overall goal of this research is to design, simulate, and verify a point-supported radial skeleton spherical robot. The motivation behind this research is to reduce the weight and increase the speed of a traditional radial skeleton spherical robot design by minimizing the number of

actuators required, using a unique geometry, and taking advantage of the dynamics. Typically, in radial skeleton spherical robots such as the Buckybot (Grande et al. 2011) and the Spiny Multipedal Robot (Nozaki et al. 2018) not all of the prismatic joints are used at once to make a discrete roll, so fewer motors could be used in a way to actuate the robot. Minimizing the required actuators in conjunction with being point-supported can increase the speed of the robot compared to its counterparts with actuators on all joints because a much more powerful actuator can be used for propulsion so momentum can be gained more easily.

Chapter 2. Overall Design

The overall design shown in Figure 1 takes advantage of the dynamics and geometry by using rounded bumpers on all faces and vertices of an icosahedron to emulate a sphere more than traditional designs like the Buckybot (Grande et al. 2011) the Spiney Multipedal Robot (Nozaki et al. 2018) and the Mochibot (Nozaki et al. 2019). The design presented in this work uses a powerful internal linear actuator to propel the robot in the desired direction.

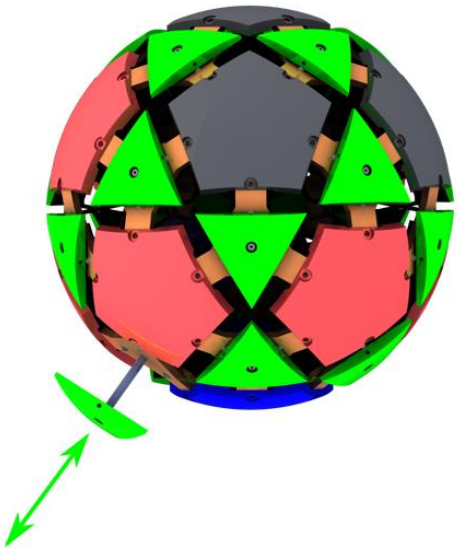

Figure 1: Overall design that shows rounded bumpers in green, black, red, and blue. The green bumpers can move radially in and outward to propel the robot in the desired direction shown by the bidirectional arrow.

2.1 Design Requirements

The design requirements for this spherical robot are shown in the table below:

Design Requirement	Goal
Average Speed	40 cm/s
Geometry	Less than 60 cm Diameter
Mass	10 kg
Number of Actuators	Use Less than 5 actuators

Table 1: Design Requirements

The average speed design requirement was determined from the Spiny Multipedal Robot (Nozaki et al. 2018) which could move at an average speed of 28 cm/s. The size requirement makes it possible for the robot to navigate in an indoor environment. A diameter of less than 70 cm also simplifies assembly and transport. The 10 kg design requirement also allows for easy transport and deployability.

To meet the design requirements, three main designs were considered and compared with the traditional design of having actuators on all joints. All the designs that were considered have an icosahedron frame made up of triangular plates connected by 138-degree angle brackets as shown in the figure below.

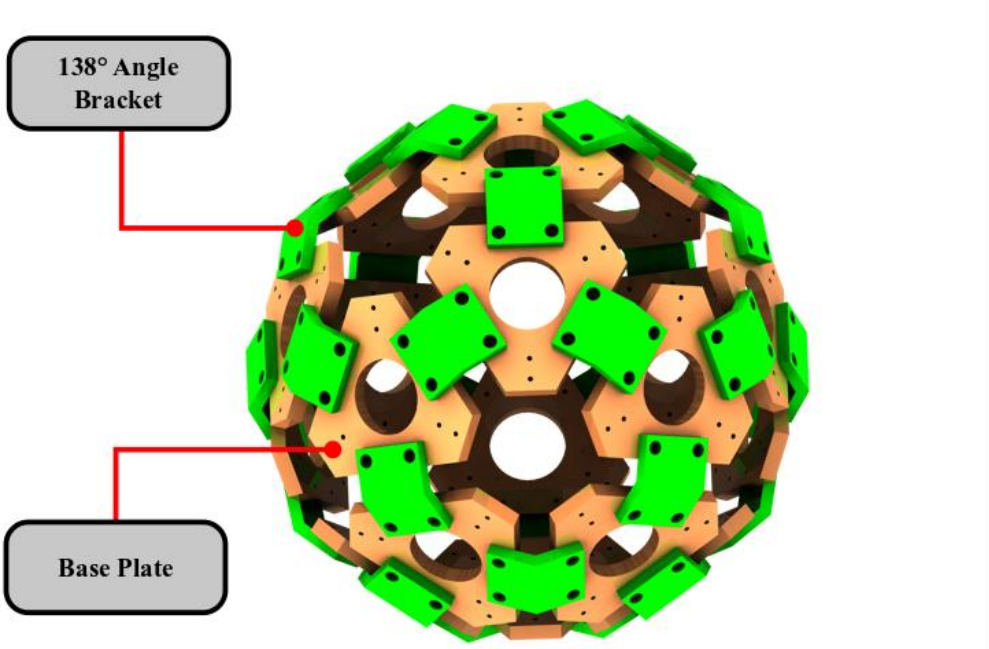

Figure 2: Icosahedron frame made up of 20 base plates and 30 138-degree angle brackets.

The traditional design in this case would be 20 lead screws each powered by its own individual motor. The most complex design that was considered used several bevel gears inside of the frame powered by a single motor with a series of electromagnetic clutches to transmit power from a bevel gear to a lead screw on each face. Another design that was considered used an internal robotic arm with a toothed clutch as an end effector to position itself to rotate lead screws on each face. The final design that was considered also used an internal arm, but instead of a lead screw on each face it used one lead screw on the internal arm that would push on spring-loaded passive joints. A pugh chart was used to help determine the best design as shown in the table below.

	Number of Actuators	Complexity	Difficulty interfacing with each joint	Control Difficulty	Totals
Weighting	3	2	2	3	
Motor for Each Joint	0	0	0	0	0
Bevel Gears and Clutches	-1	-1	0	0	-5
Internal Arm and Clutches	+1	+1	-1	-1	0
Internal Arm and single lead screw	+1	+1	-0.5	-0.5	2.5

Table 2: Pugh chart for design selection.

The design that was decided upon was the internal arm with a single lead screw design because it is much simpler than the bevel gear design and does not have the difficulty of interfacing the toothed clutches that the internal arm with clutches design has.

2.2 Major Components

The three major components of this design are the icosahedron frame, the internal arm, and the passive spring-loaded prismatic joints with triangular bumpers. The spring loaded joints have triangular rounded bumpers that form a 36 cm diameter sphere along with stationary pentagonal bumpers as shown in Figure 1. The Buckybot (Grande et al. 2011) also has rounded faces so that the robot does not fall on the actuated faces. In contrast, the design presented here has rounded bumpers on all faces and vertices to gain more momentum than polyhedron face supported designs because it does not have to stop in between steps. One of the prismatic joints is shown in Figure 3. The icosahedron frame determines two major aspects of the robot. It dictates the physical size that the motors can be and the distance that the robot travels after a discrete roll. The base plates that make up the icosahedron frame serve four purposes. The main purpose is to interface with the 138-degree angle brackets so that the entire icosahedron frame can be constructed.

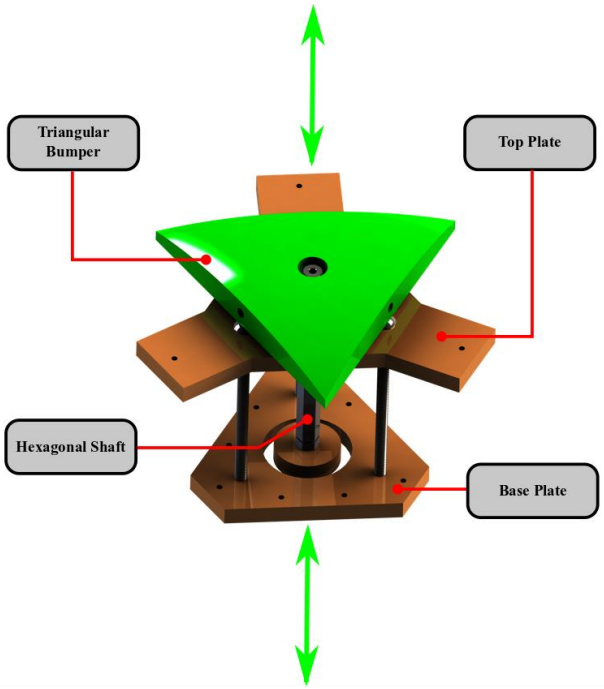

Figure 3: Prismatic joint with triangular rounded bumper. Upper bidirectional arrow shows linear motion of the joint and lower bidirectional arrow shows where the linear actuator pushes on the joint. The hexagonal shaft allows linear motion through a hexagonal hole in the top plate but restricts rotational motion. 3 Rubber bands (Not shown) will be used to spring load the joint. Certain models in this figure were obtained from McMaster-Carr and Servocity. See part numbers in the appendix.

The second purpose of the base plates is to interface with the vertex bases mounted on the vertices of the icosahedron frame shown in Figure 4 below. The base plates also have a hole in the center to allow the linear actuator on the internal arm to push an individual prismatic joint. The last role of the base plates is to connect to the prismatic joint top plates. The top plate shown in Figure 3 serves three purposes. The top plates mount to the base plates, connect to the pentagonal bumpers, and act as a linear bearing for the prismatic joint. The spring-loaded prismatic joints are made up of the top plate, hexagonal shaft, base plate, and triangular bumper.

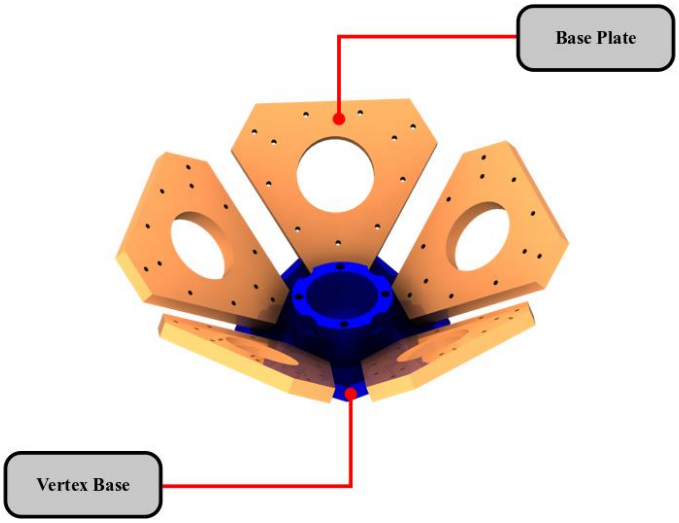

Figure 4: This figure shows the connection between the base plates and a vertex base.

The internal actuating arm, shown in Figure 5, has 3 joints. Two are revolute joints and the third is a prismatic joint. The prismatic joint is a lead screw system as discussed in section 2.1. The function of the revolute joints is to position the prismatic joint of the internal arm radially outward from the center of the icosahedron frame so that it can push on the spring-loaded prismatic joints through the hole in the center of the triangular base plates. The revolute joint motors were selected based on geometry so that they would fit in the vertex bases and inside the icosahedron frame.

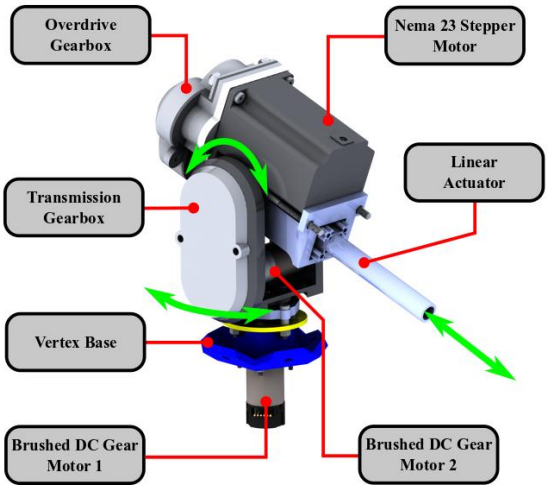

Figure 5: Internal arm showing possible motion and labeled components. The overdrive gearbox has a 1:3.2 gear ratio to transmit power from the NEMA 23 Stepper motor to the lead screw. The transmission gearbox has a 1:1 gear ratio. Bidirectional arrows show possible motion. Certain models in this figure were obtained from McMaster-Carr, Pololu, and Servocity. See part numbers in the appendix.

The motors were also sized based on the desired speed of the robot overall. When the robot travels at 40 cm/s it takes approximately 0.5 seconds to roll from one pentagonal bumper on the floor to another pentagonal bumper. The travel time from one pentagonal bumper to another is important because with a pentagonal bumper on the floor there are five potential directions that triangular bumpers can propel the robot. This means that 0.5 seconds is the time that the revolute joint motors would have to align the internal arm with a prismatic joint. The maximum rotation for the revolute joints are 180 degrees for the motor labeled "Brushed DC Gear Motor 1" in Figure 5 and 105.24 degrees for the motor labeled "Brushed DC Gear Motor 2" in Figure 5.

A trapezoidal velocity profile for the revolute joints was assumed with a total travel time of 0.5 seconds for the maximum travel rotation of 180 degrees for motor 1 and 105.24 degrees for motor 2. In the "trapveltraj" MATLAB function, the default profile sets the peak velocity at 1.5 times the average velocity. The average velocities of each motor for the maximum travel rotations are 60 RPM (rotations per minute) for motor 1 and 35 RPM for motor 2. This corresponds to maximum velocities of 90 RPM and 52.5 RPM respectively. The angular accelerations of these velocity profiles are 56.5 rad/s^2 for motor 1 and 33.0 rad/s^2 for motor 2.

Using the angular accelerations and moments of inertia the torque required to accelerate the mass of the internal arm can be found. The moment of inertia of the mass that motor 1 rotates is approximately 25.1 kg cm^2 according to a Solidworks measurement. The moment of inertia of the mass that motor 2 rotates is 31.1 kg cm^2 according to a Solidworks measurement. This results in torque requirements of 0.142 Nm for motor 1 and 0.102 Nm for motor 2.

The motors that were selected for these joints both had a no-load speed of 100 RPM and a stall torque of 2.84 Nm. The motor for the prismatic joint of the internal arm was sized based on kinematic requirements for overall motion of the robot, which required the design of an overdrive gearbox to transmit power from the motor to the lead screw while increasing speed. Each push or extension of the joint was assumed to take 0.1 seconds to be significantly less than the time it takes to roll from one pentagonal bumper to another. The travel distance of this prismatic joint is 7.5 cm.

Assuming a trapezoidal velocity profile for this prismatic joint the average velocity of the joint would be 75 cm/s, and like the calculations for the revolute joints the maximum velocity would be 1.5 times the average velocity. The maximum velocity of the internal arm prismatic joint would be 112.5 cm/s. The lead screw of the internal arm prismatic joint has an 8mm lead. Combining the maximum velocity of the prismatic joint and the 8mm lead results in the rotational velocity of 8440 RPM. The maximum RPM of the NEMA 23 stepper motor that was selected for this design is approximately 3900 RPM at 24 V half stepping with 4 A per phase. The McMaster-Carr part number of this motor is included in the appendix. The overdrive gearbox has a gear ratio of 1:3.2, so the maximum RPM of overdrive gearbox output to the lead screw is 12,480 RPM, which is higher than the 8440 RPM requirement. The dynamics of the linear actuator of the internal arm will be further discussed in section 3.1.

Chapter 3. Dynamics and Controls

3.1 System Dynamics

This robot was modeled to estimate the force requirements for the linear actuator and determine a method for controlling the robot. The robot is modeled as a 36 cm diameter sphere with an external force acting radially inward on the surface of the sphere along the axis that a prismatic joint would lie. Figure 6 shows a free body diagram of the simplified model.

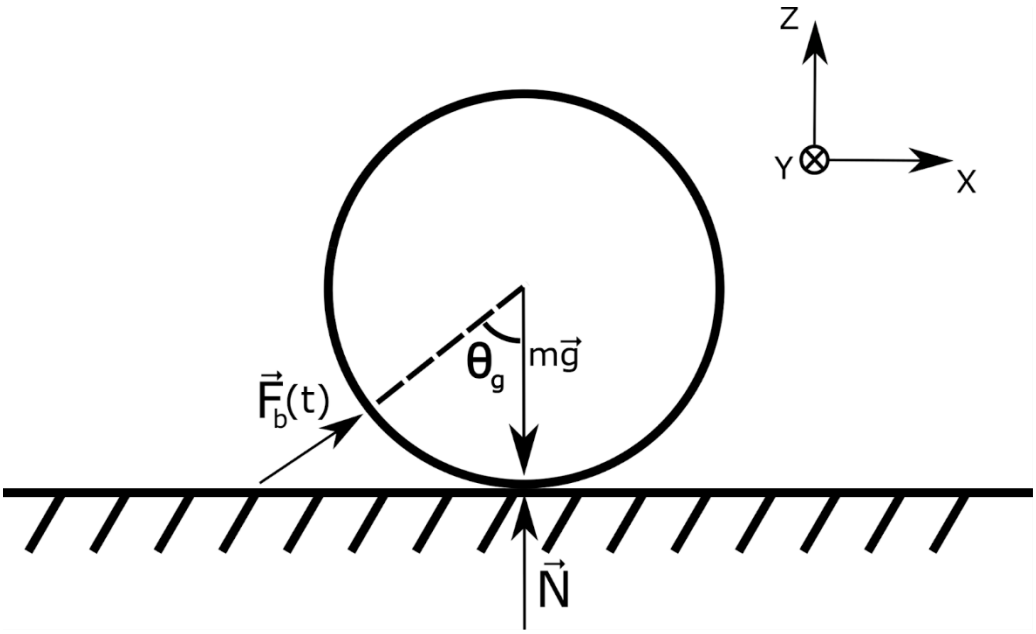

Figure 6: Free body diagram of simplified spherical model. $\vec{F}_b(t)$ is the force versus time that a prismatic joint would apply on the robot. Where $m\vec{g}$ represents the weight of the robot and $\vec{N}$ is the normal force that the floor applies on the robot. θ_g is the angle from the gravity vector to the applied bumper force.

This model makes the following simplifying assumptions: the bumper force is constant, the angle from the bumper force to the vertical stays constant while the bumper force is applied, the sphere slips on the surface while the bumper force is applied, the bumper force is small enough so that there is no vertical motion, friction is small relative to the bumper force, and the sphere rolls and does not slip when the bumper force is finished acting. Because this model has both a change in momentum and a force versus time, the impulse momentum theorem can be used to determine the necessary force to achieve the desired overall robot velocity. The estimate for the internal arm linear actuator extension of 0.1 seconds can be used to estimate the amount of time that the bumper force is applied. With the estimated time that the force is applied, the change in momentum from rest to 40 cm/s, which will give the typical average force necessary for a nominal speed of 40 cm/s. The following equations help to determine this range of average forces:

$$\vec{F}_{Net}(t)\Delta t = m(\vec{v}_2 - \vec{v}_1) \quad (1)$$

$$\overrightarrow{||\vec{F}_{Net}(t)||} = ||\vec{F}_b(t)|| \cdot \sin(\theta_g) \tag{2}$$

$$||\vec{F}_b(t)|| \cdot \sin(\theta_g) \cdot \Delta t = ||m(\vec{v}_2 - \vec{v}_1)|| \tag{3}$$

$$||\vec{F}_b(t)|| = \frac{||m(\vec{v}_2 - \vec{v}_1)||}{\sin(\theta_g) \cdot \Delta t} \tag{4}$$

Where $\vec{F}_{Net}(t)$ is the net force on the sphere, Δt is the amount of time that the bumper force is applied, m is the mass of the robot, $\vec{v}_1$ is the velocity of the robot before the force is applied, $\vec{v}_2$ is the velocity of the robot after the force is applied, $\vec{F}_b(t)$ is the force that a prismatic joint applies on the rest of the robot when actuated, and θ_g is the angle between a prismatic joint axis and gravity. The resulting bumper force is 66 N for accelerating to 40 cm/s from rest. The ranges of forces and translational speeds of the lead screw are based on a linear approximation of the torque speed curve of the selected stepper motor and the gear ratio of 1:3.2 of the overdrive gearbox. These ranges correspond to a maximum speed of the linear actuator of approximately 1.6 m/s. Based on the equation obtained from (vCalc 2016) a range of forces that the linear actuator can output is 27 N at 1.6 m/s to 151 N at 0.32 m/s. This assumes a coefficient of friction of 0.2 in the calculation of the input torque to lead screw force. The bumper force necessary to accelerate the robot from rest falls within the linear actuator force range, so this simplified model supports the selected stepper motor as sufficient to control the robot.

3.2 Control System

The control system for the robot is shown in Figure 7. The input is a waypoint that the robot is commanded to navigate to, and the outputs of the plant are the robot position, orientation, and velocity. To simplify the simulation the output from the controller will be a selected bumper to actuate, a force for the selected bumper to apply, and a duration of time to apply the force. The feedback into the system will be the position, orientation, and velocity of the robot which would be reported by an IMU (Inertial Measurement Unit) in a physical implementation, but for simulation purposes a transform sensor Simulink block will be used to determine the translational velocity.

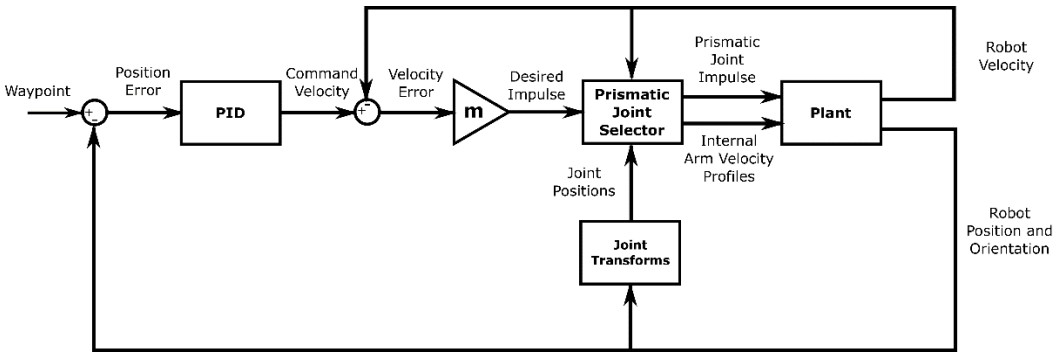

Figure 7: Control diagram, where m is the total mass.

The control system uses a PID controller to control the position of the robot as it follows a commanded trajectory. The prismatic joint selector shown in Figure 7 selects the bumper that aligns best with the opposite direction of the desired impulse vector and aligns the internal arm with it to push on the selected prismatic joint. The plant in the control diagram consists of the dynamics of the bumpers and internal arm. The control system implementation in simulation will be further discussed in chapter 4.

Chapter 4. Simulation and Experimental Results

4.1 Primitive Sphere and External Force Simulation

The primary goals of these simulations are to verify the design and tune the PID controller. The simulations discussed in this work were performed using Simulink. The first simulation that was implemented utilized a primitive sphere model available in Simulink. This simulation was used to verify the basic dynamics of the robot after one propulsion. This primitive sphere model used a diameter of 36 cm and a mass of 10 kg. Rigid transforms were used to simulate the positions of one of the triangular bumpers. An external force was added to the follower frame of the triangular bumper transform, and it applied a constant force for a finite time to simplify the simulation. To achieve a final velocity of 40.85 cm/s from rest a force of 90 N was applied for 0.1 seconds. This shows that the model developed in section 3.1 has limitations due to some of the simplifying assumptions, but 90 N is in the range of forces that the linear actuator can apply.

4.2 Pentagonal and Triangular Bumpers Simulation

The next simulation used the models of the triangular and pentagonal bumpers as shown in Figure 1. Rigid transformations from an unactuated 6 degree of freedom joint were made to prismatic joints and triangular bumpers were added for each joint. Rigid transformations from the 6 degree of freedom joint were also made to the pentagonal bumper positions. This simulation excluded the dynamics of the internal arm by making the internal arm mass zero. Additional mass was added to the robot to compensate for the mass lost from the internal arm. The motion of the internal arm was still implemented in this simulation to verify the timing between propulsions. The main purpose of this simulation was to test the robot with modeled bumpers instead of an external force on a sphere in section 4.1. The secondary purpose of this simulation was to compare it to a simulation with the internal arm dynamics included. To select a prismatic joint to actuate, a prismatic joint selector function was developed to take a desired impulse, robot velocity, and joint positions as inputs and output revolute joint positions based on trapezoidal velocity profiles using the “trapveltraj” MATLAB function to align with a selected prismatic joint that will apply a force for 0.1 seconds. This prismatic joint selector function corresponds to the prismatic joint selector block in the control diagram. The prismatic joint positions were determined from transform sensor Simulink blocks. The transform sensor blocks measured the position vector from the center of the robot to the prismatic joint positions in world frame coordinates. The positions of the revolute joints were measured from the revolute joint Simulink blocks. The prismatic joint selector function MATLAB implementation is included in the appendix. The prismatic joint selector function follows this set of steps:

1. Determine the vectors from the center of the robot to the position of each prismatic joint in fixed world frame coordinates. This step corresponds to the joint positions input to the prismatic joint selector shown in Figure 7.
2. Determine the rotational velocity axis by finding the cross product of the vertical vector and the robot velocity vector.
3. Calculate the potential travel time it would take to get to each prismatic joint by calculating the triangular velocity profiles it would take to rotate to each prismatic joint of both

revolute joints. The revolute joints move concurrently while rotating to a prismatic joint, so the revolute joint with the longer actuation time is the overall potential travel time.
4. Multiply the travel time by 2 to account for the variability in speed.
5. Use the travel times, radius of the robot, and velocity of the robot to find the angles that the robot would rotate to after each potential travel time.
6. Determine rotation matrices from the angles that the robot would roll after the potential travel times and the rotational velocity axis.
7. Multiply a rotation matrix by the corresponding bumper position to find the predicted position of one bumper after its travel time. Repeat this for all bumpers
8. Determine the triangular bumpers that are eligible to push the robot by finding their angles to the gravity vector.
9. Find the maximum absolute angle within a range of -180 to 180 degrees between the horizontal components of each of the triangular bumper vectors eligible to push and the desired impulse vector to determine the best suited bumper to push.
10. Determine the necessary force to push the triangular bumper to get the desired impulse using equation 4.
11. Divide the maximum absolute angle between the horizontal components of the selected triangular bumper vector
12. Calculate the velocity profiles for the revolute joints to move to the selected bumper and send this command to the internal arm model. This step corresponds to the internal arm velocity profiles output of the prismatic joint selector.
13. After the internal arm aligns with the selected bumper, output a constant force for 0.1 seconds to the selected prismatic joint. This step corresponds to bumper impulse output of the prismatic joint selector block in Figure 7.
14. Repeat steps 1 through 13.

The prismatic joint blocks in Simulink can be actuated by providing a linear position, or an actuator force. To simplify the simulation and ensure the limit on the force that could be applied, the prismatic joint blocks were actuated with a force. The plant block in Figure 7 corresponds to the 6 degree of freedom joint, the prismatic joints for each triangular bumper model, the pentagonal bumper models, and spatial contact between each bumper and the simulated flat surface. The plant also includes the dynamics of the internal arm, but in this section the internal arm dynamics of the plant is not implemented.

The PID block in Figure 7 was implemented using a discrete PID Simulink block. The PID parameters were tuned using the Ziegler-Nichols method as a starting point (Ellis 2016), and final gains are listed in table 3 below. The average speeds of the selected motors for the revolute joints were too slow to achieve a reasonable trajectory. This required increasing the average speed of the revolute joints to 5 times the average speed of the selected motors for these revolute joints. These results show that there was an underestimation of the motor kinematics during motor sizing.

PID Gain	Value
Proportional	2.52
Integral	0.868
Derivative	0.8641

Table 3: Gains for PID controller used in simulation without the internal arm dynamics.

A circular trajectory was traversed by the simulated robot to measure performance. The trajectory was commanded with a circular trajectory generator function. The circular trajectory generator function is included in the appendix. Plots of the trajectories of the center of the robot, command and response plots, and position errors vs. time are shown in the figures below similar to the plots in (Liu Y. et al 2008) and (Palacín J. et al. 2021).

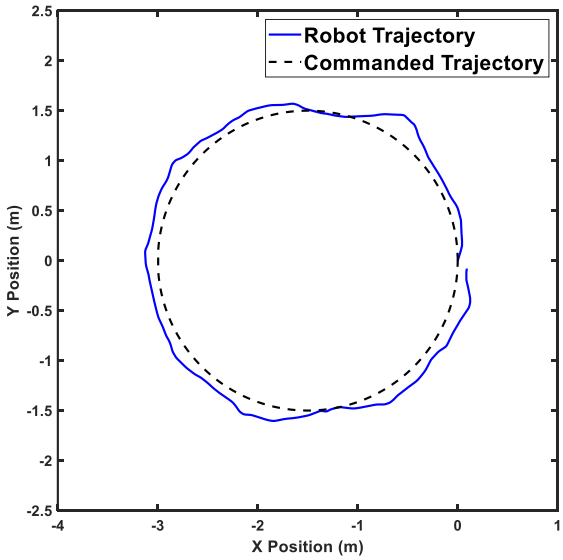

Figure 8: Plot of circular trajectory of center of robot without the internal arm dynamics.

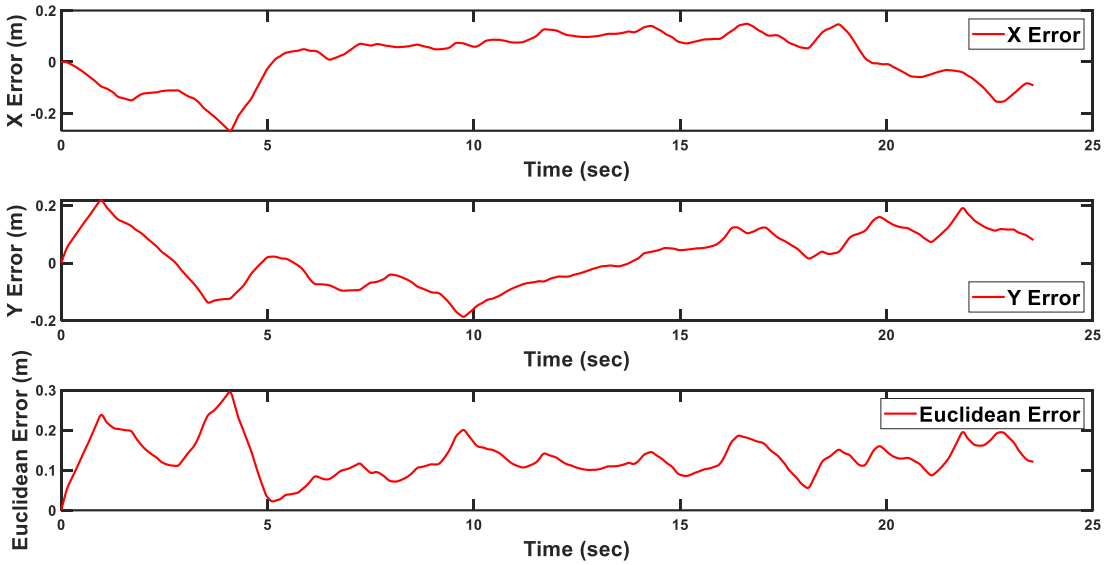

Figure 9: The x, y, and Euclidean position error versus time for the circular trajectory without the internal arm dynamics. The maximum errors in x and y were 0.2678 meters and 0.2181 meters, respectively. The Euclidean error is the square root of the sum of the x error squared and the y error squared. The maximum Euclidean error was 0.2951 meters.

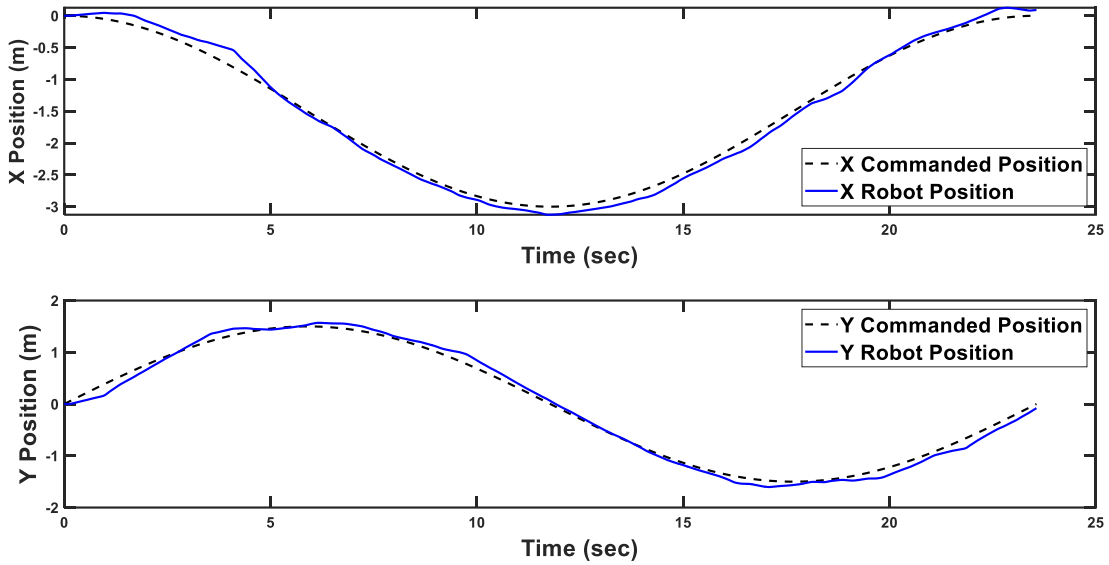

Figure 10: These plots show the x and y components of position versus time of the commanded trajectory and the actual robot trajectory without the internal arm dynamics.

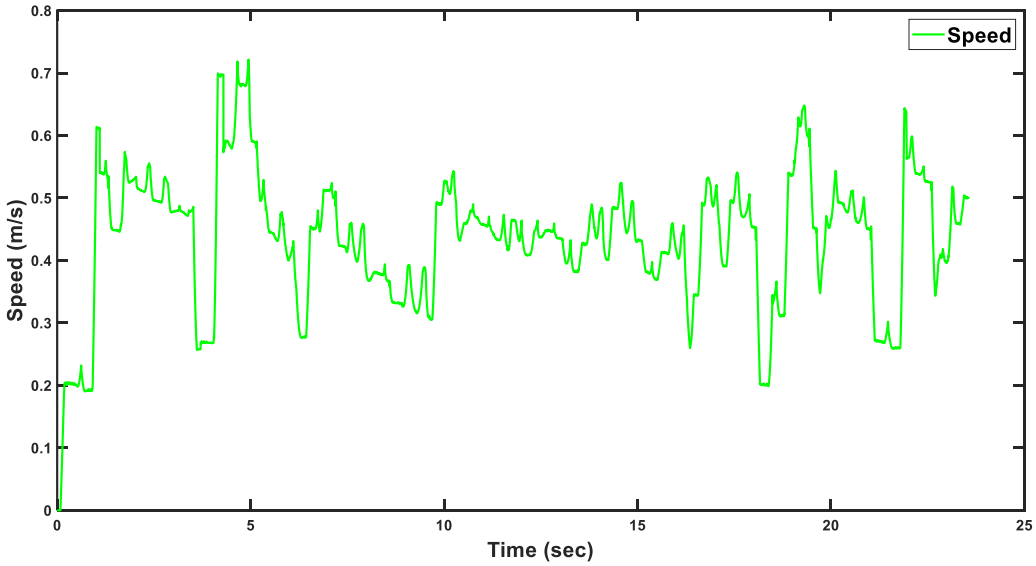

Figure 11: Speed versus time plot for circular trajectory without internal arm dynamics. The average speed of this data is 44.26 cm/s.

Figure 8 shows a plot of both the commanded and actual trajectory of the center of the robot. Figure 9 shows the x, y, and Euclidean position errors versus time. The maximum errors are all less than the diameter of the robot. The command and response plot shown in Figure 10 depicts how well the robot followed the trajectory over time, and it shows very little lag between the commanded and actual position. Figure 11 shows a plot of the speed of the robot versus time. The plot of speed versus time shows that the speed is significantly variable due to the delay

between propulsions. This simulation confirmed that the rounded bumpers can propel the robot over a circular commanded trajectory with error less than the diameter of the robot. This simulation also showed that the robot is capable of a 40 cm/s speed while following a circular trajectory when the internal arm dynamics are not included.

4.3 Internal Arm Dynamics Simulation

The internal arm dynamics adds a complication to the system because it applies reaction torque and force to the rest of the robot structure. The simulation described in this section simulates the reaction torque and force by giving the internal arm mass in its Simulink model. The results of this simulation were used to compare to the previous section to determine if the reaction torque and force had any effect on the trajectory performance. The same PID gains in table 3 were used to determine if the reaction torque and force influenced performance. The plots in Figures 12-15 below are the results of the simulated robot following a circular trajectory using the PID gains from Table 3.

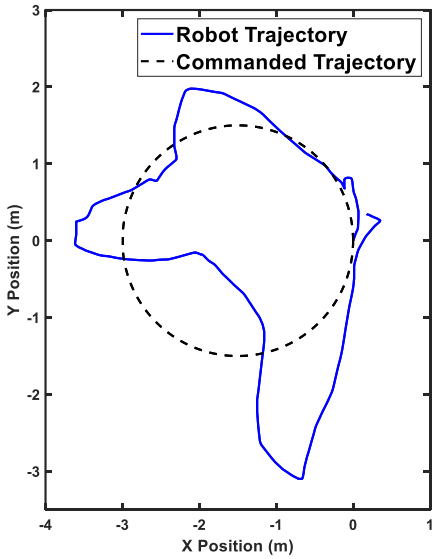

Figure 12: Plot of circular trajectory with internal arm dynamics, using PID gains from Table 3.

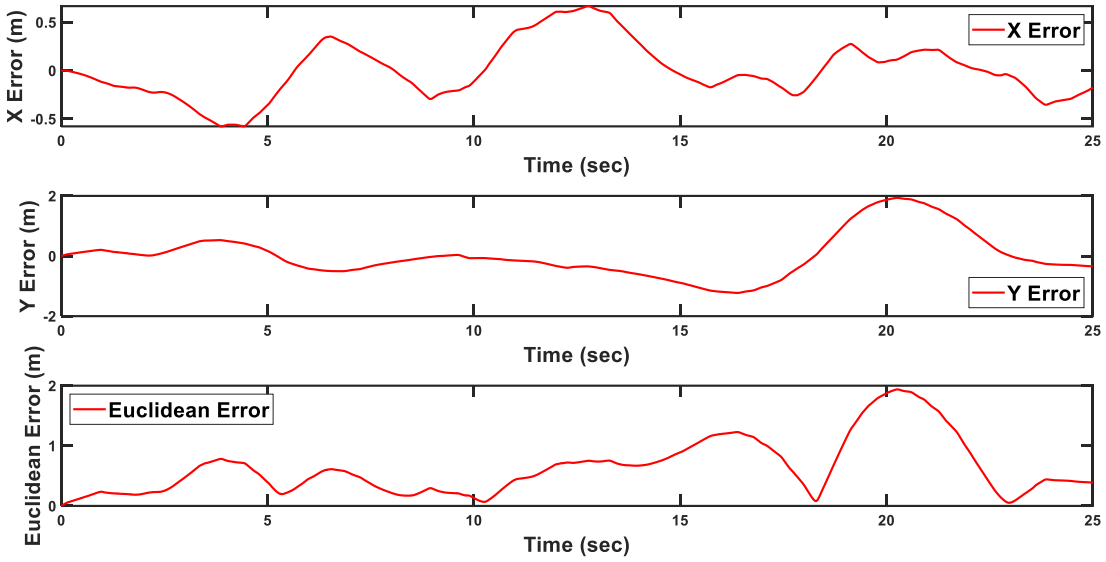

Figure 13: The x, y, and Euclidean position error versus time for the circular trajectory with the internal arm dynamics and the PID gains from Table 3. The maximum errors in x and y were 0.6669 meters and 1.9345 meters, respectively. The maximum Euclidean error was 1.9380 meters.

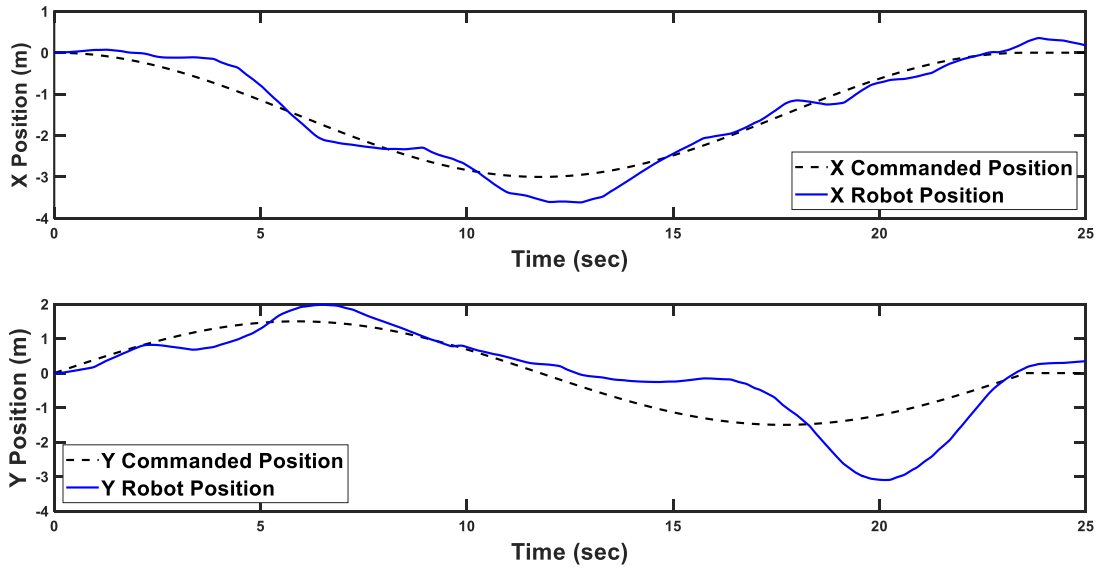

Figure 14: These plots show the x and y components of position versus time of the commanded trajectory and the actual robot trajectory with the internal arm dynamics and PID gains from Table 3.

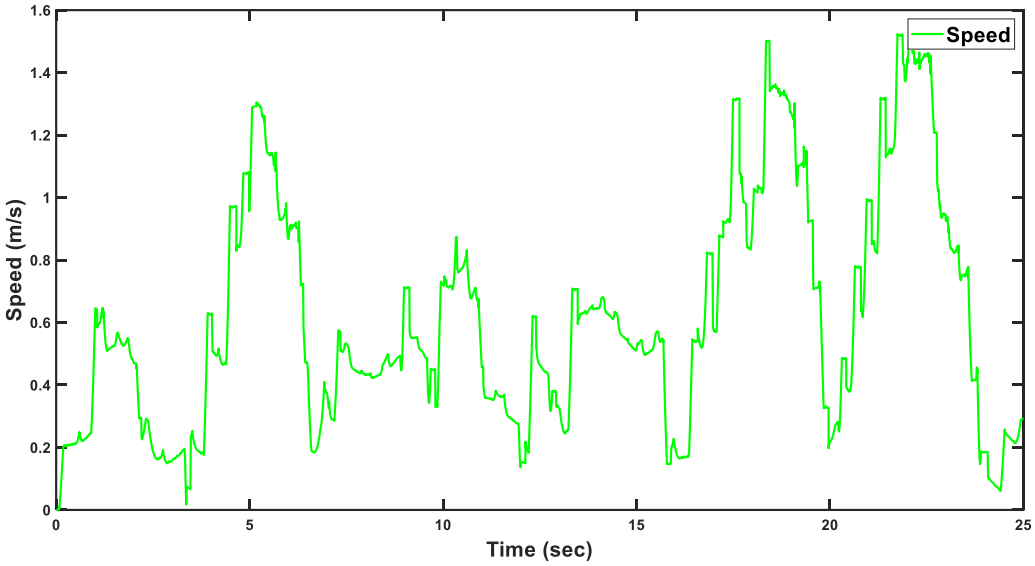

Figure 15: Speed versus time plot of circular trajectory with internal arm dynamics. The average speed was 66.55 cm/s.

These plots support the claim that the internal arm reaction torque and force make the robot much less accurate under the same conditions and same PID gains because with the same PID gains the maximum trajectory errors were much higher. The robot was tuned again to account for the reaction torque and force because the reaction torque and force were not included in the dynamics model. The following PID gains in Table 4 were used to achieve a smoother trajectory.

PID Gain	Value
Proportional	3.24
Integral	0.56
Derivative	0.6696

Table 4: PID gains for simulated robot with internal arm dynamics after re-tuning.

The following plots in Figures 16-19 show the results of the re-tuning to account for the dynamics of the internal arm.

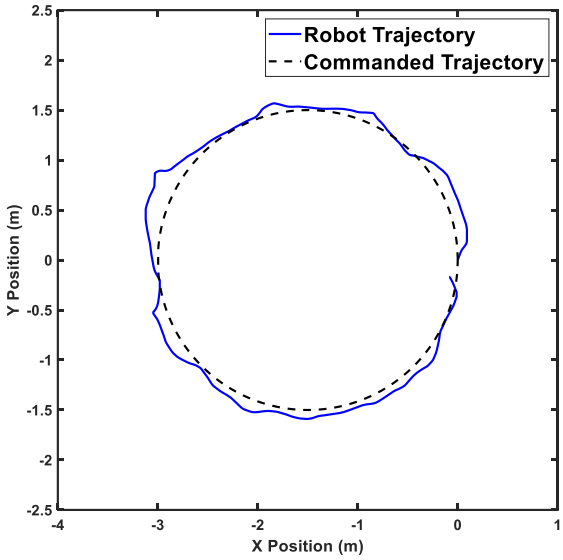

Figure 16: Plot of circular trajectory with internal arm dynamics, using PID gains from Table 4.

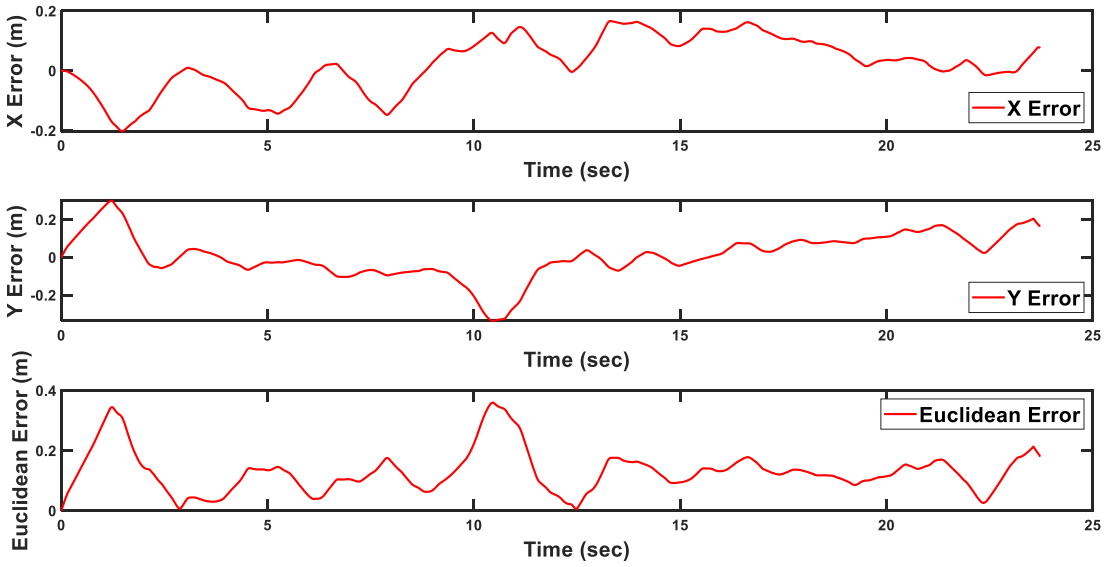

Figure 17: The x, y, and Euclidean position error versus time for the circular trajectory with the internal arm dynamics and the PID gains from Table 4. The maximum errors in x and y were 0.2020 meters and 0.3371 meters, respectively. The maximum Euclidean error was 0.3596 meters.

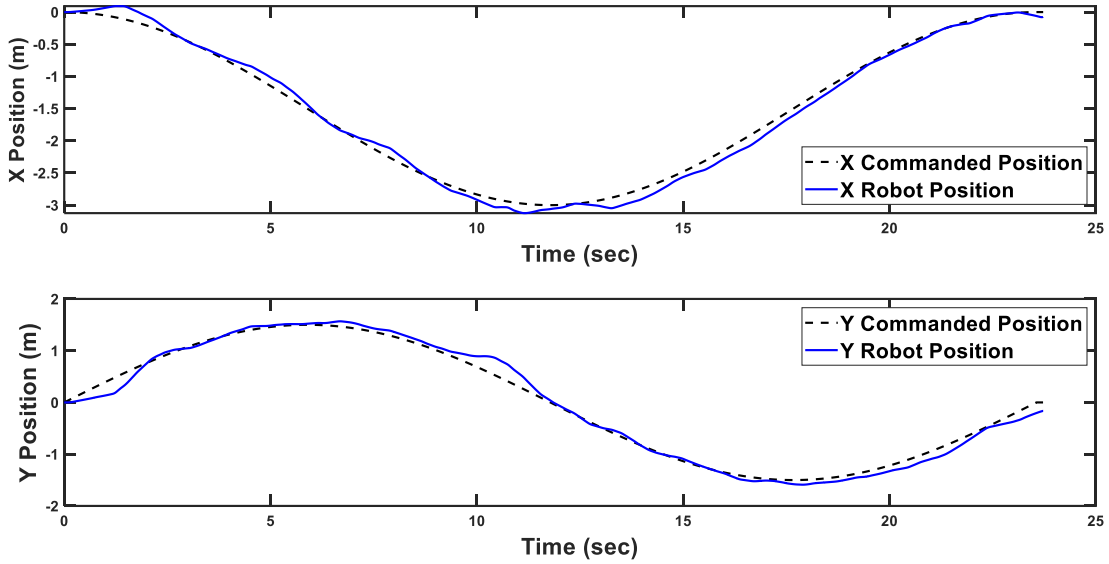

Figure 18: These plots show the x and y components of position versus time of the commanded trajectory and the actual robot trajectory with the internal arm dynamics and PID gains from Table 4.

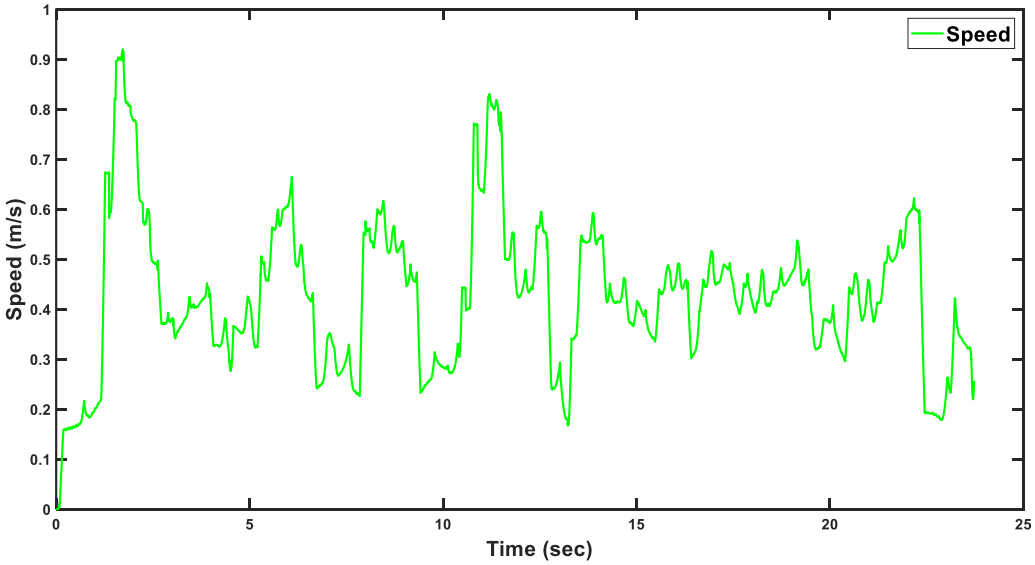

Figure 19: Speed versus time plot of circular trajectory with internal arm dynamics. The average speed was 44.62 cm/s.

These plots show that the complete Simulink model of the robot including the dynamics of the bumpers and internal arm can follow a circular trajectory with error still under the diameter of the robot in simulation.

4.4 Triangular Bumpers and Internal Arm Simulation

To assess the performance of the robot when it is not point-supported, the simulation described in this section did not include the pentagonal bumpers. This simulation did implement the internal arm dynamics. 3 cm diameter spheres were used as bumpers instead of the triangular bumpers to keep the robot from rolling out of control after a discrete roll. The prismatic joint selector function and Simulink model was slightly altered to simulate the robot without pentagonal bumpers. The PID block was removed, and the velocity feedback was also removed because they were not necessary for the control of the robot in this configuration. The main feedback was the position of the robot and an input trajectory. The plots in Figures 20 through 23 below show the results of this simulation.

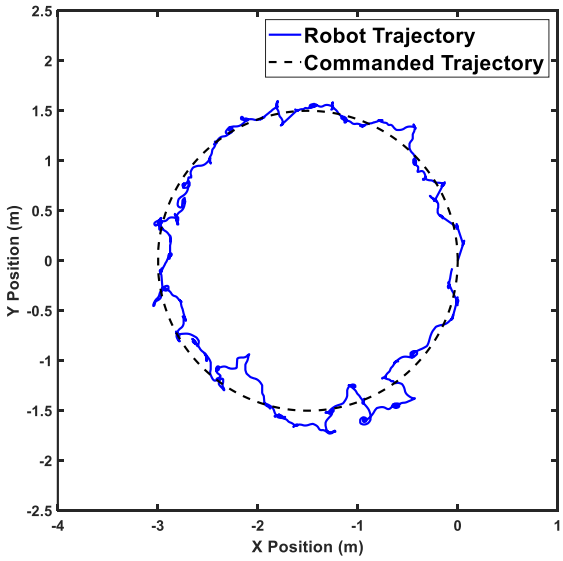

Figure 20: Plot of 10cm/s circular trajectory without pentagonal bumpers. The discrete steps can be seen in the zig-zag pattern of the trajectory.

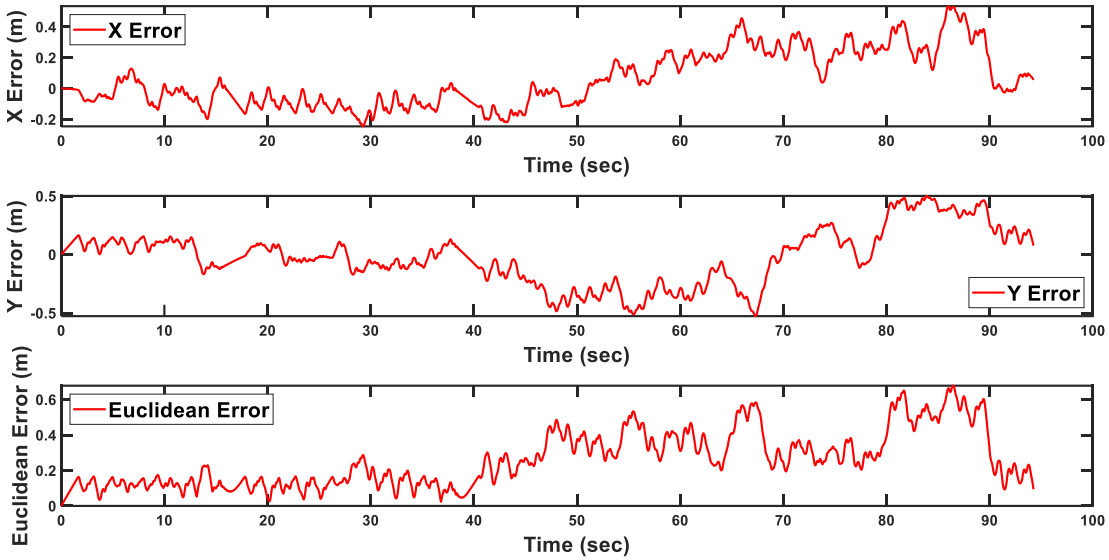

Figure 21: The x, y, and Euclidean position error versus time for the circular trajectory without pentagonal bumpers. The maximum errors in x and y were 0.5341 meters and 0.5270 meters, respectively. The maximum Euclidean error was 0.6781 meters.

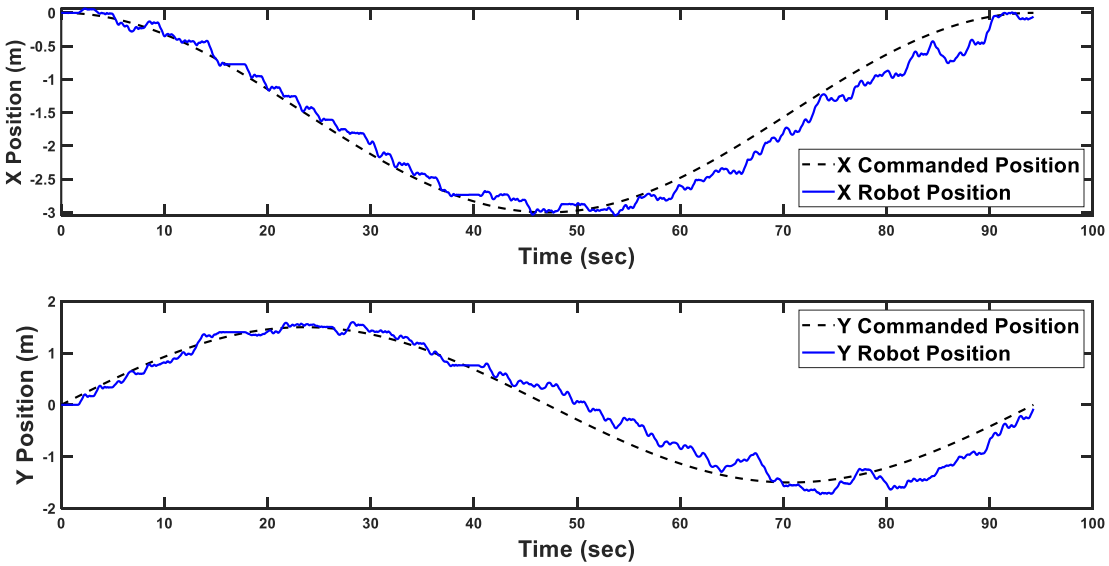

Figure 22: These plots show the x and y components of position versus time of the 10 cm/s commanded trajectory and the actual robot trajectory without the pentagonal bumpers.

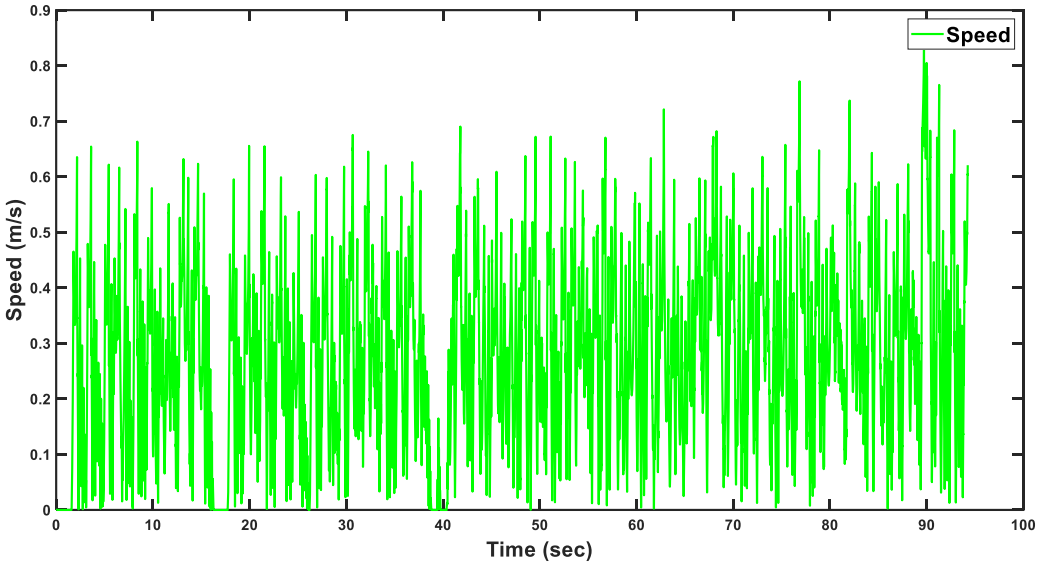

Figure 23: Speed versus time plot of circular trajectory with internal arm dynamics. The average speed was 30.07 cm/s.

These plots support the claim that being point-supported can increase the average speed of a radial skeleton spherical robot because compared to the point-supported simulations with the pentagonal bumpers, this simulated robot was only able to follow a 10 cm/s commanded trajectory. The damping coefficients between the spherical bumpers and the floor were increased to reduce the time it took for the robot to reduce its speed between propulsions. The static and kinetic friction coefficients were also increased to simulate the robot in this configuration because friction plays a larger role when the robot is not point-supported.

Chapter 5. Conclusion

This thesis has investigated the hypothesis that a point-supported discrete rolling radial skeleton spherical robot is capable of higher speeds than polyhedron face supported spherical robots in simulation. Although the simulation of the design with both triangular bumpers, pentagonal bumpers, and the internal arm dynamics showed that the high instability, and high variability in speed makes the point-supported design difficult to control. The results in section 4.4 showed that the unique actuation design for the design presented in this work does not increase the average speed of a radial skeleton spherical robot from the speed recorded by the Spiney Multipedal Robot of 28 cm/s (Nozaki et al. 2018). The unique actuation can achieve higher speeds when point-supported because it is easier to gain angular momentum with a rolling sphere compared to a polyhedron. This shows that the reduced number of actuators had a significant impact on the speed of the point-supported deformable spherical robot because higher forces could be applied to joints due to the more powerful actuator, which resulted in a higher attainable average velocity.

The performance of the robot declined when the internal arm dynamics was included in the simulation with the PID gains in Table 3 because of the reaction torque and force applied by the internal arm onto the rest of the robot. Although by re-tuning the PID gains, similar performance to the simulation without internal arm dynamics was achieved. Simulation has merit for early verification of dynamics and controls development, but it will not necessarily capture the exact behavior of an actual robot. To further verify the simulations in this thesis the robot would need to be developed and tested.

Acknowledgements

Thank you to Professor David Hardt for his guidance and support throughout this process.

Appendix

Robot Parameter	Value
Mass	10 kg
Overall Diameter	36 cm
Triangular bumper Side Length	9.8 cm
Pentagonal Bumper Side Length	9 cm
Prismatic Joint Travel Distance	6 cm
Linear Actuator Travel Distance	8 cm
Overdrive Gearbox Gear Ratio	1:3.2
Transmission Gearbox Gear Ratio	1:1
Brushed DC Gear Motor No-Load Speed	100 RPM
Brushed DC Gear Motor Stall Torque	2.84 Nm
Stepper Motor Max Speed	3900 RPM
Stepper Motor Max Holding Torque	1.67 Nm

Table 5: List of parameters of the robot.

Part	Quantity	Distributer	Part Number
M4 14mm Low-Profile Screw	20	McMaster-Carr	92855A847
M4 80mm Screw	60	McMaster-Carr	92095A173
8mm Rex Shaft 72mm	20	Servocity	2106-4008-0720

Table 6: List of parts with part numbers to reference 3D models used in the prismatic joint model shown in Figure 4.

Part	Quantity	Distributer	Part Number
Stepper Motor	1	McMaster-Carr	6627T54
10-32 1” Low-Profile Socket Head Screw	2	McMaster-Carr	92220A176
10-32 4” Pan Head Phillips Screw	1	McMaster-Carr	91772A850
10-32 5” Pan Head Phillips Screw	1	McMaster-Carr	90272A386
10-32 Locknut	2	McMaster-Carr	90633A411
M4 Low-Profile Socket Head Screw	4	McMaster-Carr	93070A107
M4 Spacer	4	McMaster-Carr	94669A006
M4 Locknut	4	McMaster-Carr	90576A103
M3 35 mm Screw	4	McMaster-Carr	92467A484
M3 Locknut	4	McMaster-Carr	90576A102
Aluminum Tube for Lead Screw Nut	1	Servocity	4101-1014-0300
120mm goRAIL Extrusion	1	Servocity	1109-0024-0120
32mm Bore Pillow Block Bearing	1	Servocity	1603-0032-0032
99:1 Gearmotor HP 12V with Encoder	2	Pololu	4847

Table 7: List of parts with part numbers to reference 3D models used in the internal arm model shown in Figure 4.

Prismatic Joint Selector Function

```
function [prismatic_joint_forces, joint_20_position_output,
joint_21_position_output] = controller(time, bumper_positions,
robot_velocity, desired_impulse, joint_20_position_input,
joint_21_position_input)
    %joints 20 and 21 refer to the internal arm revolute joints

    %angles that the revolute joints of the internal arm must
move to to
    %align with each bumper.
    joint_20_angles = [-36, -108, -180,-252,-324,0,-36,-72,-108,-
144,-180,-216,-252,-288,-324,0,-72,-144,-216,-288];
    joint_21_angles = [52.62, 52.62, 52.62, 52.62, 52.62, -10.81,
10.81, -10.81, 10.81, -10.81, 10.81, -10.81, 10.81, -10.81,
10.81, -52.62, -52.62, -52.62, -52.62, -52.62];

    %set revolute joint velocities
    joint_20_position_output = joint_20_position_input;
    joint_21_position_output = joint_21_position_input;

    %joint_22_travel_time is the amount of time that the force is
applied
    %by a prismatic joint
    joint_22_travel_time = 0.1;

    %robot parameter
    overall_radius_of_robot = 0.18;

    %a threshold for robot velocity to know when to actuate
joints and when
    %to make certain calculations. In meters/sec
    min_robot_velocity = 0.05;

    %the maximum and minimum angles to gravity that a bumper can
be to be
    %considered to be actuated
    max_angle_to_gravity = 40;
```

```
    min_angle_to_gravity = 30;

    %initialize the prismatic joint forces with negative 1 newton
to keep
    %them in the closed position
    holding_force = 1;
    prismatic_joint_forces = -holding_force*ones(20, 1);

    %the peak rotational acceleration of both revolute joints in
deg/sec^2
    revolute_joints_acc = 51572;

    %the discrete sample time of this MATLAB function block
    sample_time = 0.001;

    %persistent variables

    %start_actuation_time is the time that the joints should
start moving
    persistent start_actuation_time;

    %stage_of_operation dictates if the function is determining a
bumper to actuate determining the
    %required force, moving the internal arm, or actuating a
bumper.
    persistent stage_of_operation;

    %force_output is the calculated force that a prismatic joint
will apply
    %from the input parameters
    persistent force_output;

    %this variable is the index of the bumper that will be
actuated
    persistent selected_bumper_to_actuate_index_persistent;

    %trajectories of each revolute joint
    persistent joint_20_trajectory;
    persistent joint_21_trajectory;
```

```
    %timing for each revolute joint travel
    persistent joint_20_time;
    persistent joint_21_time;

    %the angles from bumpers to gravity vector after each
potential
    %internal arm overall travel time
    persistent predicted_angles_to_gravity;
    persistent max_angle_between_impulse_and_bumper;

    %initialize the persistent variables
    if isempty(start_actuation_time)
        start_actuation_time = 0;
    end

    if isempty(stage_of_operation)
        stage_of_operation = 0;
    end

    if isempty(force_output)
        force_output = -5;
    end

    if isempty(selected_bumper_to_actuate_index_persistent)
        selected_bumper_to_actuate_index_persistent = 0;
    end

    if isempty(joint_20_trajectory)
        joint_20_trajectory = 0;
    end

    if isempty(joint_21_trajectory)
        joint_21_trajectory = 0;
    end

    if isempty(joint_20_time)
        joint_20_time = 0;
    end
```

```
    if isempty(joint_21_time)
        joint_21_time = 0;
    end

    if isempty(predicted_angles_to_gravity)
        predicted_angles_to_gravity = zeros(1, 20);
    end

    if isempty(max_angle_between_impulse_and_bumper)
        max_angle_between_impulse_and_bumper = 0;
    end

    %stage 1 selects a bumper to actuate
    if stage_of_operation == 0 && time > sample_time
        %rotational_velocity_axis is a unit vector that
represents the rotational
        %axis of the entire robot as it rolls
        if norm(robot_velocity) > min_robot_velocity
            rotational_velocity_axis = cross([0 0 1],
robot_velocity);
            rotational_velocity_axis =
rotational_velocity_axis/norm(rotational_velocity_axis);
        else
            rotational_velocity_axis = zeros(3, 1);
        end

        %these joints follow trapezoidal velocity profiles to
rotate to a bumper and
        %the travel time it takes to get to each bumper is
calculated based on the angular acceleration
        potential_joint_20_displacements = 180 - joint_20_angles
- joint_20_position_input;

        for i = 1:length(potential_joint_20_displacements)
            if potential_joint_20_displacements(i) > 180
                potential_joint_20_displacements(i) =
potential_joint_20_displacements(i) - 360;
            elseif potential_joint_20_displacements(i) < -180
```

```
                potential_joint_20_displacements(i) =
potential_joint_20_displacements(i) + 360;
            end
        end

        potential_joint_20_travel_times =
sqrt(abs(2*potential_joint_20_displacements)/revolute_joints_acc)
;

        potential_joint_21_displacements = -joint_21_angles -
joint_21_position_input;

        for i = 1:length(potential_joint_21_displacements)
            if potential_joint_21_displacements(i) > 180
                potential_joint_21_displacements(i) =
potential_joint_21_displacements(i) - 360;
            elseif potential_joint_21_displacements(i) < -180
                potential_joint_21_displacements(i) =
potential_joint_21_displacements(i) + 360;
            end
        end

        potential_joint_21_travel_times =
sqrt(abs(2*potential_joint_21_displacements)/revolute_joints_acc)
;

        %potential travel times finds the maximum travel time
between
        %both joints because they move at the same time so
whichever has
        %the highest travel time will be the overall travel time
of the
        %move to the next bumper
        %the coefficient of 2 ensures that the selected bumper
accounts for
        %the variability in velocity
        potential_travel_times =
max(potential_joint_20_travel_times ,
potential_joint_21_travel_times)*2;
```

```
        %using potential travel times, the angle that the robot
would roll
        %after each move to a bumper can be found

        if norm(robot_velocity) > min_robot_velocity
            potential_travel_angle_for_each_chosen_bumper =
(norm(robot_velocity) *
potential_travel_times)/overall_radius_of_robot;
        else
            potential_travel_angle_for_each_chosen_bumper =
zeros(1, 20);
        end

        %initialize predicted_bumper_positions
        %predicted_bumper_vectors are the bumper positions after
the
        %internal arm would move to the corresponding bumper
        predicted_bumper_positions = zeros(3, 20);

        for i = 1:20
            %The rotation matrix corresponding to the rotational
velocity
            %axis and the travel angle is found for each bumper
            TheoraticalRollRotationMatrix =
rotationVectorToMatrix(-
potential_travel_angle_for_each_chosen_bumper(i) *
rotational_velocity_axis);

            predicted_bumper_positions(:, i) =
TheoraticalRollRotationMatrix * bumper_positions(:, i);
        end

        %find the predicted angles to gravity
        predicted_angles_to_gravity = real(acosd(max( min([0 0 -
1]*
predicted_bumper_positions./vecnorm(predicted_bumper_positions) ,
1), -1)));
        predicted_angles_to_gravity_thresholds =
```

```
(predicted_angles_to_gravity <
max_angle_to_gravity).*(predicted_angles_to_gravity >
min_angle_to_gravity);

        %find the angles from the desired impulse vector to each
bumper vector
        %only using the horizontal components of the predicted
bumper vectors
        predicted_vectors_to_bumpers_with_zero_z = cat(1,
predicted_bumper_positions(1:2, :), zeros(1, 20));

        angles_between_x_axis_and_bumpers =
real(acosd(max(min((([1 0 0] *
predicted_vectors_to_bumpers_with_zero_z))./vecnorm(predicted_vec
tors_to_bumpers_with_zero_z), 1), -1))) .*
sign(predicted_vectors_to_bumpers_with_zero_z(2, :));
        angle_between_x_axis_and_desired_impulse =
real(acosd(max(min((dot([1 0 0],
desired_impulse))/norm(desired_impulse), 1), -
1)))*sign(desired_impulse(2));
        angles_between_bumpers_and_desired_impulse =
angles_between_x_axis_and_bumpers -
angle_between_x_axis_and_desired_impulse;

        for
i=1:length(angles_between_bumpers_and_desired_impulse)
            if angles_between_bumpers_and_desired_impulse(i) >
180
                angles_between_bumpers_and_desired_impulse(i) =
angles_between_bumpers_and_desired_impulse(i) - 360;
            elseif angles_between_bumpers_and_desired_impulse(i)
< -180
                angles_between_bumpers_and_desired_impulse(i) =
angles_between_bumpers_and_desired_impulse(i) + 360;
            end
        end

        %find the maximum angle and its index between the desired
impulse and horizontal
```

```
        %components of the bumper positions
        [max_angle_between_impulse_and_bumper,
selected_bumper_to_actuate_index_persistent] =
max(abs(angles_between_bumpers_and_desired_impulse).*predicted_an
gles_to_gravity_thresholds);

        %ensure that a bumper was actually selected and use a
threshold for
        %the maximum angle between impulse and the selected
bumper
        if selected_bumper_to_actuate_index_persistent > 0 &&
max_angle_between_impulse_and_bumper > 135
            stage_of_operation = 1;
        end
    %stage 1 sets the selected bumper, the force to be applied,
the
    %start_actuation time, and the joint trajectories
    elseif stage_of_operation == 1
        stage_of_operation = 2;
        start_actuation_time = time;

        force_output = min(
(abs(norm(desired_impulse))/joint_22_travel_time)/abs(sin(deg2rad
(predicted_angles_to_gravity(selected_bumper_to_actuate_index_per
sistent)))), 170);
        force_output = force_output *
(max_angle_between_impulse_and_bumper/180);

        joint_20_displacement = (180 -
joint_20_angles(selected_bumper_to_actuate_index_persistent)) -
joint_20_position_input;

        %ensure that the joint displacements are within [-180,
180]
        if joint_20_displacement > 180
            joint_20_displacement = joint_20_displacement - 360;
        elseif joint_20_displacement < -180
            joint_20_displacement = joint_20_displacement + 360;
        end
```

```
        %find the joint 20 travel time
        joint_20_travel_time =
sqrt(abs(2*joint_20_displacement)/revolute_joints_acc);

        %if the joint travel time is less than the sample time
then the
        %displacement is also small so the joint is already in
the
        %right position
        if joint_20_travel_time > sample_time
            [joint_20_trajectory, ~, ~, joint_20_time] =
trapveltraj([joint_20_position_input (joint_20_position_input +
joint_20_displacement)], floor(joint_20_travel_time/sample_time),
'Acceleration', revolute_joints_acc);
        else
            joint_20_trajectory = joint_20_position_input;
        end

        joint_21_displacement = ( -
joint_21_angles(selected_bumper_to_actuate_index_persistent)) -
joint_21_position_input;

        %ensure that the joint displacements are within [-180,
180]
        if joint_21_displacement > 180
            joint_21_displacement = joint_21_displacement - 360;
        elseif joint_21_displacement < -180
            joint_21_displacement = joint_21_displacement + 360;
        end

        %find the joint 21 travel time
        joint_21_travel_time =
sqrt(abs(2*joint_21_displacement)/revolute_joints_acc);

        %if the joint travel time is less than the sample time
then the
        %displacement is also small so the joint is already in
the
```

```
        %right position
        if joint_21_travel_time > sample_time
            [joint_21_trajectory, ~, ~, joint_21_time] =
trapveltraj([joint_21_position_input (joint_21_position_input +
joint_21_displacement)], floor(joint_21_travel_time/sample_time),
'Acceleration', revolute_joints_acc);
        else
            joint_21_trajectory = joint_21_position_input;
        end
    %stage 2 corresponds to the movement of the revolute joints
of the
    %internal arm
    elseif stage_of_operation == 2
        [~,joint_20_trajectory_index] = (min(abs((time -
start_actuation_time) - joint_20_time)));

        if joint_20_trajectory_index < 1
            joint_20_trajectory_index = 1;
        elseif joint_20_trajectory_index >
length(joint_20_trajectory)
            joint_20_trajectory_index =
length(joint_20_trajectory);
        end

        [~,joint_21_trajectory_index] = (min(abs((time -
start_actuation_time) - joint_21_time)));

        if joint_21_trajectory_index < 1
        elseif joint_21_trajectory_index >
length(joint_21_trajectory)
            joint_21_trajectory_index =
length(joint_21_trajectory);
        end

        %when both movements have concluded move on to the next
stage
        if joint_20_trajectory_index ==
length(joint_20_trajectory) && joint_21_trajectory_index ==
length(joint_21_trajectory)
```

```
            stage_of_operation = 3;
            start_actuation_time = time;
        else
            %assign the joint position outputs to the trapezoidal
trajectories
            joint_20_position_output =
joint_20_trajectory(joint_20_trajectory_index);
            joint_21_position_output =
joint_21_trajectory(joint_21_trajectory_index);
        end

    %stage 3 corresponds to the movement of the selected
prismatic joint
    elseif stage_of_operation == 3
        %stage 3 outputs a constant force for the
joint_22_travel_time ,
        %and then closes by applying a negative force. It also
makes sure
        %that the bumper has actually closed using the bumper
positions
        joint_22_start_time = start_actuation_time;

        if (time - joint_22_start_time) < joint_22_travel_time &&
(time - joint_22_start_time) > 0

prismatic_joint_forces(selected_bumper_to_actuate_index_persisten
t) = force_output;
        elseif (time - joint_22_start_time) >=
joint_22_travel_time && (norm(bumper_positions(:,
selected_bumper_to_actuate_index_persistent)) >
overall_radius_of_robot-0.01)

prismatic_joint_forces(selected_bumper_to_actuate_index_persisten
t) = -holding_force;
        elseif (time - joint_22_start_time) >=
joint_22_travel_time && norm(bumper_positions(:,
selected_bumper_to_actuate_index_persistent)) <
overall_radius_of_robot
            stage_of_operation = 0;
```

```
        end
    end
end
```

Published with MATLAB® R2020a

Circular Trajectory Generator Function

```
function [waypoint, end_time] =
circular_trajectory_generator(time)
    %radius of circular trajectory
    radius = 1.5;

    %the speed that the robot should move at to follow the
trajectory
    desired_trajectory_speed = 0.1;

    %the time it takes to complete the trajectory at the desired
speed
    end_time = (2*pi*radius)/desired_trajectory_speed;

    %set the waypoint output position
    if (time < end_time)
        waypoint =
[(radius*cos(desired_trajectory_speed/radius*time) - radius)
(radius*sin(desired_trajectory_speed/radius*time)) 0];
    else
        waypoint = [0 0 0];
    end

end
```

Published with MATLAB® R2020a

Bibliography

Chase, R., Pandya, A. “A Review of Active Mechanical Driving Principles of Spherical Robots.” *Robotics*, 22 Nov. 2012 https://www.researchgate.net/publication/277684319_A_Review_of_Active_Mechanical_Driving_Principles_of_Spherical_Robots

Ellis G., “Control System Design Guide (Fourth Edition)”, *Butterworth-Heinemann*, 2 Sep. 2016

Grande, R. C., et al. “Buckybot: Preliminary Control and Mapping Algorithms for a Robot Geometrically Based on a Truncated Icosahedron.” *Johns Hopkins APL Technical Digest, 2013.*

Liu Y., et al., “Omni-directional mobile robot controller based on trajectory linearization.” *Robotics and Autonomous Systems*, 31 May 2008, *https://www.sciencedirect.com/science/article/pii/S0921889007001431?casa_token=YWBd1kKKUQgAAAAA:fpknGaGI70OybVwSgRfSfIahUWQKR3E4tzFELLUf6V8-RhQHo5hnaBhyqMSir6O26L9hvwlA*

Leadscrew Torque (lift). *vCalc*. Mar 15, 2016. https://www.vcalc.com/wiki/vCollections/Leadscrew+Torque+(lift)

Nozaki, H. et al. “Continuous Shape Changing Locomotion of 32-legged Spherical Robot.” *IEEE*.7 Jan. 2019.

Nozaki, H. et al. “Shape changing locomotion by spiny multipedal robot/” *IEEE*. 26 Mar. 2018.

Palacín J. et al. “Evaluation of the Path-Tracking Accuracy of a Three-Wheeled Omnidirectional Mobile Robot Designed as a Personal Assistant.” *NCBI*. Oct. 29 2021. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8587751/