Multi-robot Path Planning for a Swarm of Robots that Can Both Fly and Drive

Brandon Araki[1], John Strang[1], Sarah Pohorecky[1], Celine Qiu[1], Tobias Naegeli[2] and Daniela Rus[1]

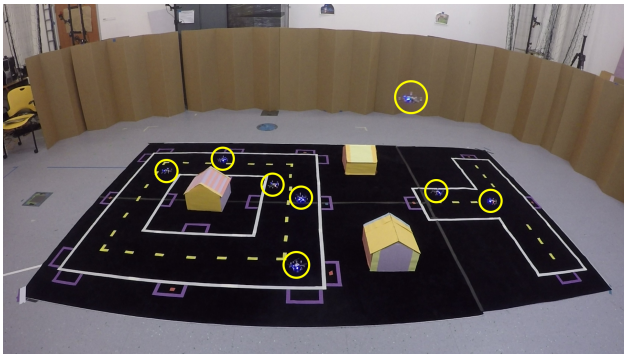
***Abstract*— The multi-robot path planning problem has been extensively studied for the cases of flying and driving vehicles. However, path planning for the case of vehicles that can both fly and drive has not yet been considered. Driving robots, while stable and energy efficient, are limited to mostly flat terrain. Quadcopters, on the other hand, are agile and highly mobile but have low energy efficiency and limited battery life. Combining a quadcopter with a driving mechanism presents a path planning challenge by enabling the selection of paths based off of both time and energy consumption. In this paper, we introduce a framework for multi-robot path planning for a swarm of flying-and-driving vehicles. By putting a lightweight driving platform on a quadcopter, we create a robust vehicle with an energy efficient driving mode and an agile flight mode. We extend two algorithms, priority planning with Safe Interval Path Planning and a multi-commodity network flow ILP, to accommodate multimodal locomotion, and we show that these algorithms can indeed plan collision-free paths for flying-and-driving vehicles on 3D graphs. Finally, we demonstrate that our system is able to plan paths and control the motions of 8 of our vehicles in a miniature town.**

I. INTRODUCTION

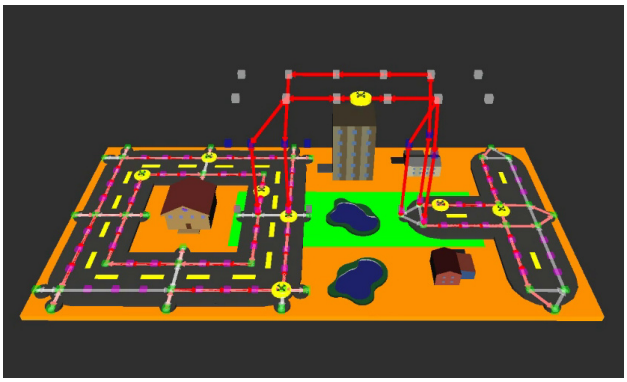
We envision a future where swarms of vehicles that can both fly and drive coordinate effectively to accomplish tasks such as pick-up-and-delivery, surveillance, exploration, and construction. We consider how a group of robots that have the ability to both move on the ground and to fly decide which locomotion modality to choose.

Flying and driving robots face two different sets of challenges - flying robots, particularly quadcopters, have limited battery life but are fast and maneuverable, whereas driving robots are energetically efficient but are constrained to move on a plane. The concept of a flying-and-driving vehicle is powerful because by combining the two forms of locomotion, one can overcome the energy concerns of flying and the spatial constraints of driving. In this paper, we accomplish this task by weighting both time and energy in the path planning cost function to strike a balance between speed (flying) and energy efficiency (driving).

The ability to move both on the ground and through the air is very common in nature - countless insects, birds, and other types of animals possess both wings and feet in order to nest, shelter, avoid aerial obstacles or bad weather, and perform many other tasks. But despite the omnipresence of terrestrial locomotion abilities in flying animals, relatively few small aerial robots are equipped with wheels. The ability to move on the ground would allow a quadcopter to navigate narrow passages, position itself for surveillance or data collection, transport heavy loads, and accomplish many other tasks it could not otherwise perform.



(a) Flying cars traversing a town



(b) The same moment in simulation

Fig. 1: Experimental and simulated flying cars

This paper presents a working system for the coordination and control of flying-and-driving robots in a city-like setting. Two multi-robot path-planning algorithms, priority planning with Safe Interval Path Planning (SIPP) and an optimal integer linear program-based algorithm, have been implemented with multimodal locomotion in mind. A small fleet of wheeled quadcopters with a novel design has been constructed. And lastly, a system for controlling these wheeled quadcopters along their desired trajectories has been implemented. The final demonstration, showing these wheeled quadcopters traveling by air and ground through a miniature town, anticipates the useful applications that

This work was supported by The National Science Foundation Grant No. 1240383.

[1]B. Araki, S. Pohorecky, J. Strang, C. Qiu, and D. Rus are with CSAIL, the Stata Center, Building 32 Vassar Street Cambridge, MA 02139 USA araki, strang, pohorecky, qiu, rus@mit.edu

[2]T. Naegeli is with the Advanced Interactive Technologies Lab, ETH Zurich, Zurich, Switzerland naegelit@student.ethz.ch

flying-and-driving vehicles will hopefully one day have.

In summary, the main contributions of this paper are:

1) a hardware platform for a miniaturized wheel-based robot that can fly and a system infrastructure for a swarm of such robots
2) two modified algorithms for multi-robot path planning that choose the locomotion modality and enable each robot to reach its goal
3) experiments in a miniaturized road environment with roads, no-fly zones, and disconnected sections that require flying.

II. RELATED WORK

The multi-robot path planning problem (MPP) has been the subject of extensive research. MPP is a challenging problem because the configuration space grows exponentially with the number of robots [1]. It can be made tractable through decentralized solutions such as [2]–[4] that delegate to each robot the responsibility for avoiding other robots. Centralized algorithms exist that can find feasible collision-free paths for multiple robots in $O(n^3)$ time or faster [5]. Priority planning is a suboptimal but fast centralized solution to MPP in which priorities are assigned to robots and paths are then found for each robot in order of its priority [6], [7]. A number of tricks exist for making priority planning better. For example, [8] suggests limiting the planning horizon, rotating priorities amongst robots, and using a true distance heuristic to get better paths from priority planning. [7] proposes the idea of well-formed infrastructures in order to generate graphs and scenarios with guaranteed solutions. Meanwhile, approaches such as [9], [10] provide optimal solutions but become intractable when the graph size or number of robots goes past a certain limit. However, these problems can be made tractable by allowing for suboptimal solutions by for example splitting the problem into multiple smaller problems in the time domain. We contribute to this literature by extending MPP to include robots with multiple modes of locomotion. We made two planners, one a fast priority planner and the other an optimal ILP, to show that the MPP can be extended to include the modified objective function of flying cars.

A handful of robots that can travel on the ground and in the air have been developed, such as the Flying Monkey [11], MALV [12], and DALER [13], among others [14]–[17]. Most of these designs place passive wheels on a quadcopter and use the rotors of the quadcopter for both aerial and terrestrial locomotion. DALER, a fixed-wing aircraft, can rotate its wings to move on the ground. MALV has wings and wheels, while the Flying Monkey places a foldable crawling mechanism on the bottom of a quadcopter. However, the design we chose placing wheels on the bottom of a quadcopter although possibly the simplest and most obvious choice for a flying-and-driving robot, seems only to have been developed commercially [17].

In [11], experiments showed that the crawling mechanism was $\sim$3x more efficient than the quadcopter used; meanwhile, in [12], the wheeled mechanism was $\sim$5x less efficient than the winged aircraft. This suggests that depending on the flying and driving mechanisms, ground locomotion may not always be more energy efficient than air locomotion. However, the existence of aerial constraints, such as no-fly zones, storms, or busy aerial traffic, as well as the convenience of taxiing on the ground for small adjustments in position, make possession of a driving mechanism a benefit for most aerial vehicles.

Lastly, the implementation of software architecture capable of controlling swarms of robots has also been the focus of much interest. Due to computational, radio, and software constraints, controlling a swarm of robots, particularly using a central computer, is a difficult task. [18]–[21] among others have succeeded in coordinating swarms of quadcopters by choosing robust robots with robust communication protocols using robust software to control them. The Crazyflie, which we used in this project, is a popular open-source quadcopter used for example in [21]–[23] that can communicate with a computer using a 2.4 GHz radio.

III. PROBLEM STATEMENT

Assumptions We assume that we have a swarm of uniform robots that can drive and fly. They have a limited energy budget and therefore energy consumption is a major factor in the path planning.

Environment We model the environment by a 3D graph that consists of a 2D road system and an aerial region, either of which may contain disconnected regions as long as the overall graph is connected. The map can accommodate static obstacles, elevated "helipads", and "no-fly zones".

Problem Given n robots with unique start and goal positions, find collision-free paths for each robot.

Solution We developed and implemented two algorithms:

1) Priority planning with Safe Interval Path Planning: a relatively fast algorithm that gives guaranteed solutions under certain conditions; however, it is not optimal.
2) Multi-commodity network flow ILP: an NP-hard problem that can provide optimal solutions in a reasonable amount of time when the number of robots and graph vertices is not too high.

IV. PROBLEM FORMULATION

Let $\mathcal{W}$ denote a workspace with $\mathcal{W} \subset \mathbb{R}^3$. This space is approximated by a connected, directed graph $G = (V, E)$. There are n robots $R = \{r_1, \ldots, r_n\}$ that can be approximated by cylinders of radius r and height h. Each robot r_i is assigned a $task$ to move from its start position s_i to a goal position g_i. A $scenario$ is defined to be a collection of tasks $\{(s_1, g_1), \ldots (s_n, g_n)\}$.

A path is a map $p_i : \mathbb{N} \to V$. A path is $satisfying$ if $p_i(0) = s_i$, $p_i(t_f) = g_i$ for some time index $t_f \in \mathbb{N}$, and for any $0 \leq t < t_f$, $\{p_i(t), p_i(t+1)\} \in E$ or $p_i(t) = p_i(t+1)$. A trajectory $\pi_i : [0, \infty) \to \mathcal{W}$ maps a time to coordinates in $\mathcal{W}$; the transformation from p_i to π_i bridges the gap between the abstract graph G and the continuous workspace $\mathcal{W}$.

The trajectories π_i, π_j of two robots are $conflict\text{-}free$ if and only if the bodies of the robots i, j never intersect when they follow the trajectories π_i and π_j.

Problem 1 (Multi-Robot Path Planning Problem)**.** *Given a workspace $\mathcal{W}$, a graph G, and a scenario S specifying the start and end points for n robots, find trajectories $\pi_1, \ldots, \pi_n$ that are $satisfying$ and $conflict\text{-}free$.*

V. METHODS

Two multi-robot path planning algorithms were implemented for this project: priority planning with Safe Interval Path Planning (SIPP) and an integer linear program (ILP) modeling the MPP problem as a multi-commodity network flow problem. We developed new objective functions and graph data structures in order to extend traditional MPP algorithms to handle the case of multimodal locomotion.

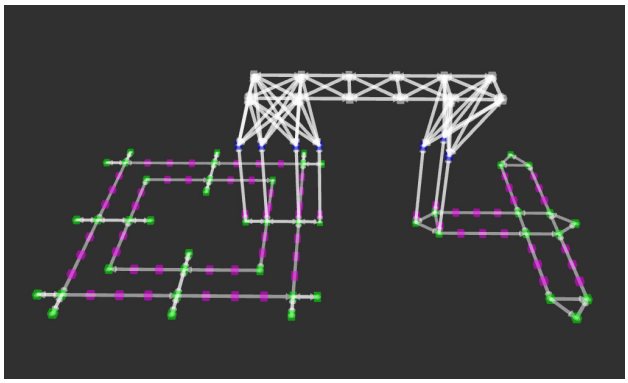


Fig. 2: Graph structure of the environment. Land nodes are green; land waypoint nodes are purples; air nodes are blue or white and connected by white edges.

A. Graph Data Structure

Graph G is defined in Section IV. Each edge e_i is assigned a length d_i that is the Euclidean distance between its two endpoints. A cost function $c(r_i, e_j, t)$, described in the next section, assigns a cost to each edge for each robot; this function also accepts self-referencing edges (v_i, v_i) and returns the cost of waiting at that node. The graph contains 3 unique types of nodes: parking nodes, land nodes, and air nodes. Parking nodes exist only on the sides of roads, where, as will be explained in the next section, they are critical for guaranteeing feasible solutions. Land nodes represent roads, and air nodes represent chunks of air where only a single flying car can be present. There is a layer of air nodes above the ground, as well as "interface" air nodes that connect the ground and air layers. The 3 node types create 4 types of edges: takeoff edges, air edges, landing edges, and ground edges. Each type of edge has a unique velocity associated with it. Fig. 2 shows the graph structure of an example map. Green node and edges on the bottom represent the road system; blue nodes and edges represent air nodes that interface with the two white air layers.

B. Objective Function

We assume two modes of locomotion: flying and driving, each with an associated power consumption, P_f and P_d, and velocity, v_f and v_d. For the purposes of this paper we assume that power consumption and velocity remain constant for each mode of locomotion. We calculate the energy cost of each edge connecting two nodes separated by a distance d; in the case of flight, we also consider the gain in potential energy during takeoff caused by lifting the mass m of the vehicle from z_1 to z_2. The work associated with flying is

$$W_f = \frac{P_f d}{v_f} + mg \cdot max(z_2 - z_1, 0) \tag{1}$$

and with driving is

$$W_d = \frac{P_d d}{v_d} \tag{2}$$

We can then calculate the cost of edge e_i with the function

$$c(e_i) = \mu \frac{W}{W_{max}} + (1 - \mu) \frac{t}{t_{max}} \tag{3}$$

where $0 \leq \mu \leq 1$ and W_{max} and t_{max} are the maximum possible energy and time of any edge in the graph. μ is a tuning parameter that allows the planner to weight energy and time in the cost function. When μ is 0, the function only weights time and the planner will simply minimize time. When μ is 1, the function only weights energy and the planner will minimize energy consumption. The ability to tune μ allows users to choose between time and energy. For example, if implemented for a flying taxi, a customer who values time over energy can pay more to make μ lower, so that the planner will value time more than energy.

C. Battery Life

We added a battery life constraint to the planning problems by estimating the current consumption over each edge. Given battery capacity C in Ah, battery voltage $V(r_i)$ as measured for robot i, average power draw $P(e_j)$, distance $d(e_j)$, and velocity $v(e_j)$, an approximate current $I(r_i, e_j)$ and current consumption $I_c(r_i, e_j)$ can be calculated by

$$I(r_i, e_j) = \frac{P(e_j)}{V(r_i)} \tag{4}$$

$$I_c(r_i, e_j) = \frac{I(r_i, e_j) d(e_j)}{v(e_j)} \tag{5}$$

In our planners, battery capacity is not allowed to fall below a specified level C_{low}.

D. Priority Planning with SIPP

Priority planning assigns a priority number to each robot under consideration and plans collision-free paths for each robot in order of its priority; as it iterates through the robots, it saves the robots' paths and the nodes within a radius r of each path to the obstacle space. Therefore priority planning is non-optimal and in a naive implementation is not guaranteed to find a solution. However, if the graph on which the robots

Algorithm 1: Safe Interval Path Planning

Algorithm SIPP
- **Input:** G, set of safe intervals, robot index i, initial state s_0, goal state s_g
- **Output:** collision-free path, updated safe intervals
- $OPEN = \emptyset$;
- insert $(s_0, c = 0, e = h(s_0), t =$ start time$)$ into $OPEN$;
- $C(s_0) = 0$;
- $T(s_0) =$ starting time;
- **while** s_g *not expanded* **do**
  - remove (s, c, e, t) with the smallest e-value from $OPEN$;
  - $successors = getSuccessors(s, t)$;
  - **foreach** s' *in successors* **do**
    - **if** s' *was not visited before* **then**
      - $C(s') = T(s') = \infty$;
    - **if** $C(s') > c + cost(s')$ ***or*** $T(s') > time(s')$ **then**
      - $C(s') = min(c + cost(s'), C(s'))$;
      - $T(s') = min(T(s'), time(s'))$;
      - insert $(s', c + cost(s'), c + cost(s') + h(s'), time(s'))$ into $OPEN$;
      - remove all tuples from $OPEN$ with $s = s'$ and $c >= C(s')$ and $t >= T(s')$
- recover p_i by backtracking from g_i;
- **return** p_i;

are traveling is structured as a well-formed infrastructure in which the robots start and finish on endpoints that do not obstruct the paths of other robots, then every robot is guaranteed to have a feasible path [7]. The notion of a well-formed infrastructure easily extends to 3D space so that a path is guaranteed as long as each flying car starts at a parking node, has an empty parking node as its endpoint, and does not travel through other parking nodes on its way to its goal.

Our Safe Interval Path Planning algorithm extends the one in [24] and finds collision-free paths for robots with multimodal locomotion by extending A* to the time domain, as outlined in Algorithm 1. Unlike an approach such as cooperative A* that discretizes the time domain [8], SIPP allows search in continuous time. It achieves this by compressing time for each node into "safe intervals" in which no obstacle passes through the given node. For example, a node through which no vehicles pass has a safe interval $(0, \infty)$. If a vehicle passes through the node from times 4.5 to 5.5, the node now has two safe intervals $(0, 4.5)$ and $(5.5, \infty)$. We define a state s to be a node-interval pair, in which a node is associated with one of its safe intervals. $T(s)$ stores the earliest-time arrival, $C(s)$ stores the lowest cost-to-come, and $h(s)$ represents the estimated cost-to-go. Arrivals in states are stored in the set $OPEN$ as tuples, so that the state s_i, the cost-to-come c, the sum of the cost-to-come and estimated cost-to-go e, and the arrival time t are associated with that arrival. $cost(s)$ returns the cost associated with state s. $state(v, i)$ returns the state associated with node v and time i, while $node(s)$ returns the node associated with state s.

Algorithm 2: Successor Function

Function *getSucccessors(s, t) : successors* **is**
- $successors = \emptyset$;
- **foreach** v' *in* $Neighbors(node(s))$ **do**
  - $\mathcal{L}$ = interval list of v';
  - dt = time to traverse $(node(s), v')$;
  - **foreach** *safe interval* i *in* $\mathcal{L}$ **do**
    - **if** $end\_time(i) > t + dt$ ***and*** *start_time(i) < end_time(interval(s))* **then**
      - t_{arr} = earliest arrival time in $\mathcal{L}$;
      - $cost = c(r_i, (node(s), v'), t_{arr} - t)$;
      - **if** v' *is the goal* ***or*** v' *is not parking* **then**
        - s' = state(v',i);
        - associate s' with time t_{arr} and $cost$;
        - insert s' into $successors$;
- **return** $successors$;

The successor function takes a state and the robot's arrival time to that state. It then finds all overlapping safe-intervals for all neighboring nodes, storing each node-safe interval pair as a successor state, along with the earliest arrival time and the cost. In [24], SIPP guarantees time-cost optimality by storing only the earliest arrival to a state, as any later arrival would simply have a sub-set of the earlier arrival's options.

In order to optimize energy as well as time, the choice of successor becomes more complicated. One cannot consider only the earliest arrival, as that may not have the lowest cost. Nor can we ignore all but the lowest cost arrival, as earlier arrivals could have more available safe intervals and thus a lower cost path overall. To maintain optimality, one would have to consider all arrivals to a state that have a lower cost than all previous arrivals, but this causes the number of successors considered to expand exponentially. We therefore sacrifice optimality (we already sacrificed multi-robot optimality by using priority planning) to speed up computation times by only considering the earliest arrival and lowest cost arrival to a state.

Once a path is found, the safe-intervals are updated. Nodes are considered unsafe for the time that the robot is within a certain radius of them. Following that, the next robot plans its path, using the updated set of safe-intervals.

E. Multi-commodity flow with ILP

Our algorithm is based off the algorithm found in [9]. In [9], the MPP is reformulated as a multi-commodity network flow problem. This is achieved by discretizing the time domain and connecting nodes in the time domain to each other

Algorithm 3: MPP-ILP-Optimization

Algorithm Model Construction
  Input: G, T_{max}
  Output: ILP model
  construct a *time-expanded* version of G by making a copy of every node for every time step and linking nodes at time t_i to nodes that are reachable at time t_{i+1};
  add arc capacity constraints;
  add flow conservation constraints;
  add scenario completion constraints;
  add meet collision constraints;
  add head-on collision constraints;
  add battery-life constraints;
  add objective function;
  return *ILP model*

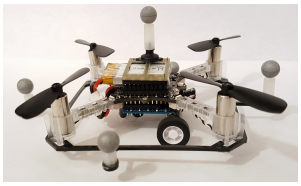
with arcs, forming a new graph G'. Multi-commodity flow can be solved by adding a binary variable x_{ij} representing the presence or absence of each robot i to each arc j, and then adding constraints that prevent robots from colliding with each other, as outlined in Algorithm 3. The specifics of the graph construction and constraints can be found in [9]; however, we added an additional battery life constraint and a new objective. Given binary variables x_{ij} and $|G'|$ arcs in G', the battery constraint for each robot i can be expressed as

$$\sum_{0 \leq j < |G'|} I_c(r_i, e_j) \cdot x_{ij} \leq C_{low} \tag{6}$$

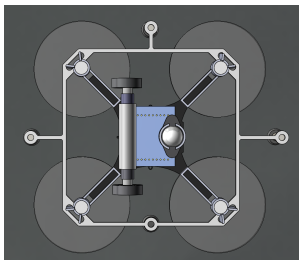
and the objective function as

$$\min \sum_{0 \leq j < |G'|, 1 \leq i \leq n} c(r_i, e_j) \cdot x_{ij} \tag{7}$$

In [9], the distance between nodes is assumed to be 1, and all of the robots move with the same velocity. In order to accommodate multiple velocities we designed a new graph structure. Given a basic graph of the road system and desired air, takeoff, landing, and ground velocities, we add "waypoint" nodes between the existing ground nodes so that the time it takes to move between nodes takes about 1 timestep. We also space out the aerial nodes so that the time it takes to travel between nodes takes about 1 timestep. In the planning step we then assume that it takes exactly 1 timestep to travel between any two adjacent nodes; this way, the velocity a robot must travel between adjacent nodes remains close to, but not exactly, the original desired velocity. Our formulation differs from the original formulation in that our objective function incorporates both time and energy; our graph contains arcs of different lengths that are calculated based off of the robots' velocities; and we include a battery voltage constraint.



(a) The Crazyflie with driving mechanism and motion capture markers



(b) A flying car from the bottom, showing the driving apparatus (two wheels and a ball caster) and the motors and propellers of the quadcopter.

Fig. 3: Mechanical design of the flying car

VI. Robot Platform

A. Mechanical Design

The mechanical design of the flying car was kept as simple as possible in order to make the fabrication of a small swarm possible. The finished robot can be seen in Fig. 3a. We used Bitcraze's Crazyflie 2.0 as the base of the system since it is fully open source and programmable. Fig. 3b shows the underside of the robot, where the driving mechanism is most visible. The driving portion of the flying car consists of a PCB with a motor driver; a carbon fiber tube with two small motors was epoxied onto the bottom of the PCB, and a ball caster was glued onto the bottom as a passive wheel element. The masses of the main components of the robot are listed in Table I. Although adding two motors to the platform increased its mass, they enabled us to use differential control for driving. This system was small, safe, and extremely robust. The firmware was modified so that the wheels could be controlled via the Crazyflie's Crazyradio command interface.

Mass Table		
Item	Mass (g)	Mass Percent
Motor Base	8.3 g	20.2%
Motion Capture Markers	4.2 g	10.2%
Crazyflie	28.5 g	69.5%
Total	41.0 g	100%

TABLE I: Masses of the elements of the flying car

B. Control

The planners return a list of waypoints and arrival times for the flying car to follow. This list of waypoints is then interpolated to make a more refined trajectory for trajectory following. For ground waypoints, a pure pursuit controller is used to follow the desired trajectory. The two ground motors allow for simple differential steering control. When a transition from a ground waypoint to an air waypoint is detected, takeoff is initiated and an LQR flight controller

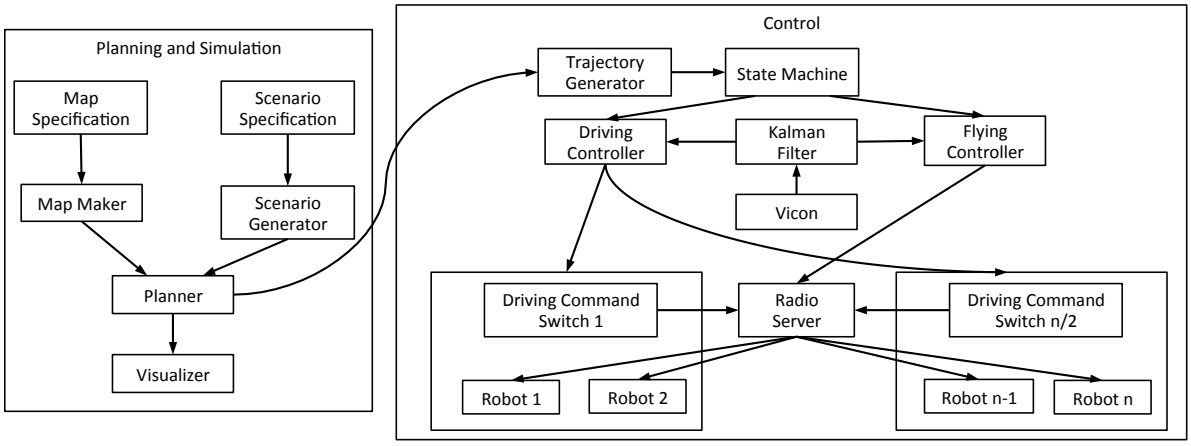


Fig. 4: System architecture

Energetics					
Vehicle	Speed (m/s)	Run Time (min)	Max Dist (m)	Ave Pow (W)	COT
Crazyflie	0.3	5.7	103	6.86	81.8
Flying Car	0.3	5.0	90	7.83	64.9
Wheels	0.1	42	252	0.60	14.9

TABLE II: Comparison of the energetics of the Crazyflie, flying car, and wheel base

takes over, sending yaw, pitch, roll and thrust commands to the Crazyflie's on-board attitude controller. When the transition from air to ground it detected, a landing sequence is triggered and the flying car lands near its target. A Kalman filter with position and yaw estimates from a Vicon motion capture system is used to estimate the state of the robots.

C. System Architecture

This project was written in the Robot Operating System (ROS) environment, and the overall architecture of the system can be seen in Figure 4. The process starts with a map configuration file, in which the road layout, as well as any no-fly zones and helipads, is specified. A "scenario specification" file, which specifies the parking nodes at which each robot will begin and end its path, can also be supplied. If not, a random scenario will be generated. The generated 3D environment and the scenario are supplied to the planner, which calculates a set of collision-free paths for the robots. The map and the paths are sent to a visualizer (such as RViz) or to a trajectory generator. The trajectory generator smooths the path into closely spaced waypoints and send them to a state machine that determines based off of the node type whether to send commands to the driving or flying controller, and if it chooses the flying controller, whether to takeoff, land, or fly. The flying controller sends flight commands directly to a radio server. We found that a single Crazyradio can support communication with at most 2 Crazyflies reliably. The driving controller uses a slower communication protocol than the flying controller, so we found that we had to serialize the driving commands being sent to a single radio. This was accomplished using a "driving command switch" that reads the driving commands for two robots and sends the commands in serial to the radio server.

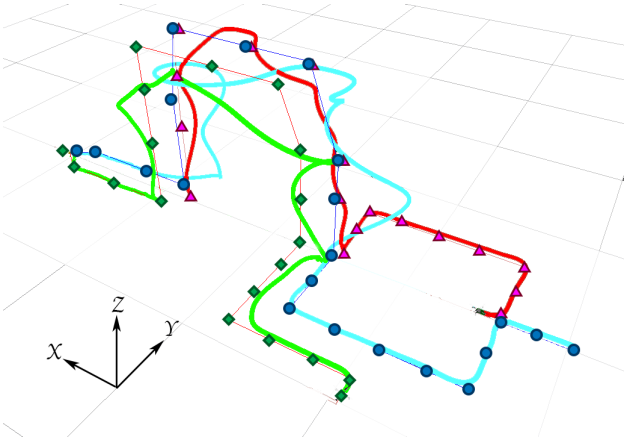


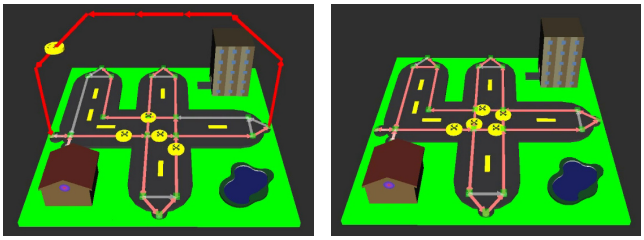
Fig. 5: Driving and flight paths of three flying cars. Bold lines are recorded trajectories and dots indicate commanded waypoints of the path with the corresponding color.

VII. EXPERIMENTS & RESULTS

A. Energetics

Experiments were conducted to determine the power consumption and battery life of the robots using different modes of locomotion. The results can be seen in Table II. Adding wheels to the base of the Crazyflie reduced its flying time by ∼12%. However, the battery life of the flying car using wheels was ∼8.4 times longer than that of the flying car when it flew, and about ∼7.4 times longer than that of the Crazyflie. This implies that at the speeds used in the simulations, the flying car could travel 90 m by flying and 252 m by driving. The cost of transport of driving is 87% lower than that of flying, corroborating the results found in

[11] and supporting our argument that flying can serve as a high-cost, high-speed transport option while driving serves as a low-cost, low-speed option.



(a) In priority planning, the lowest priority car flies over traffic

(b) The optimal planner coordinates all robots simultaneously so none have to fly

Fig. 6: Simulation experiments

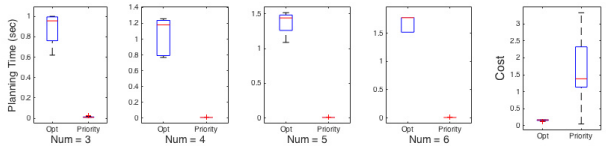


Fig. 7: Planning times vs. number of vehicles and average cost of paths

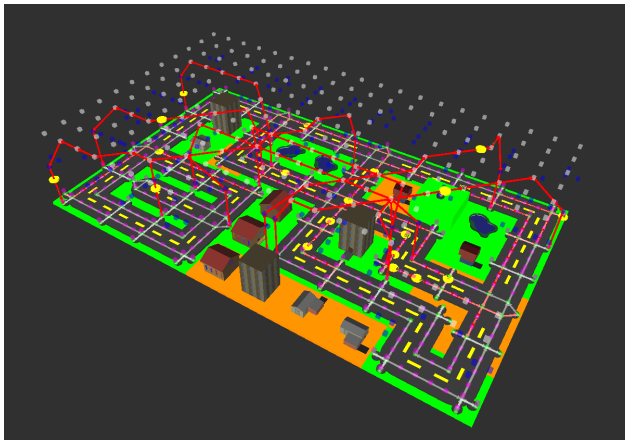


Fig. 8: Paths for 20 vehicles with flying and driving.

B. Simulation

Simulations were run to test the abilities of the system. Selected simulations are shown in the attached video. We ran simulations on over 20 maps; the largest map, shown in Fig. 8, had 934 nodes. We could plan for up to 80 vehicles in the large map; Fig. 8 shows paths for 20 vehicles. Fig. 6a shows an example of how a flying car could fly over traffic instead of waiting at an intersection for other cars to pass. Fig. 6 is also a comparison of the priority planner and the optimal planner; in Fig. 6a, the priority planner has the lowest priority car fly over the other cars in a high-cost path; in Fig. 6b, the optimal planner finds a way to coordinate the cars in order to main low-cost driving paths for all of them. Fig. 7 shows a comparison of the run times and the path cost returned by the optimal and priority-based planners. 20 trials were run for both planners for 3-6 vehicles in the 125-node map in Fig. 6. Note that planning time did not vary significantly with the number of vehicles. The plot on the right of the figure shows the average cost, which is a dimensionless value, of all 80 trials for both planners. The mean cost of the optimal planner was 0.155, or 91% lower than the 1.76 mean cost of the paths returned by the priority-based planner. Meanwhile, the mean planning time of the priority-based planner was 0.106 seconds, 0.85% of the value of the 1.25-second mean planning time of the optimal planner. Therefore the two planners offer a clear tradeoff in planning time versus cost optimality. The optimal planner is slow but returns low-cost paths, and the priority-based planner is fast but returns relatively high-cost paths.

C. Physical Experiment

Finally, a physical experiment using 8 flying cars was conducted. We provided a scenario in which each robot had to travel from a start position to a goal position. The full experiment can be viewed in the video attachment. A 4.2m x 2.1m map was constructed using velvet mats. Fig. 1a shows the experiment in progress; Fig. 1b shows the same moment in time in simulation. The map of the demonstration was constructed so that the road system consists of two disconnected regions; in order to travel from one region to another, the flying cars must fly above the gap between the two. This situation is common in the real world; for example, the gap between the road sections could be a wall, a lake, a fallen tree, or any number of other obstacles. Moreover, only a narrow corridor of land, colored in green in Fig. 1b, has available airspace; the rest of the land, colored in orange, is a no-fly zone. The narrow corridor forces the cars to coordinate their motion, and such corridors of restricted flight are also a possibility in the future; for example, delivery drones or personal flying cars may not be allowed to fly over residential areas at night. The recorded flying and driving trajectories of three of the flying cars is shown in Fig. 8. The robots successfully completed the objective of the experiment by traveling from their start positions to their goal positions in collision-free paths.

VIII. Discussion & Conclusion

In this paper, we introduced a framework for multi-robot path planning for a swarm of flying-and-driving vehicles. By putting a lightweight driving platform on a Crazyflie, we created a robust vehicle with an energy efficient driving mode and an agile flight mode. We extended two algorithms, priority planning with SIPP and a multi-commodity network flow ILP, to accommodate multimodal locomotion, and we showed that these algorithms can indeed plan collision-free paths for flying-and-driving vehicles on 3D graphs. We created a system architecture to coordinate the planning and control of a swarm of our vehicles, and we demonstrated that

our system was able to plan paths and control the motions of 8 of our vehicles in a miniature town.

Although we explored multi-robot path planning for a swarm of robots with multimodal locomotion in this paper, there are many other avenues of exploration regarding this type of robot. Path planning through an unknown environment, for example a disaster zone, with a robot that can fly and drive would enable a robot to decide when to fly over debris or drive through small holes. A surveillance robot could quickly fly to its target, use its wheels to position itself to collect data, and then fly back once its mission is complete. Flying taxis could decide when to speed through the air or drive on congested streets based on how much a customer is willing to pay. At a construction site, robots could deliver heavy loads by wheel and then fly back to pick up the next payload. By presenting a working system for multi-robot, multimodal planning and control, this work hopes to inspire further exploration of the tasks that flying-and-driving robots make possible.

ACKNOWLEDGMENT

We are grateful to Jingjin Yu for advice in implementing his multi-commodity flow ILP; to Alex Wallar for his help with ROS; to Cenk Baykal for his help with experiments; and to Robert Katszchmann for advice on the mechanical design of the flying cars.

REFERENCES

[1] John E Hopcroft, Jacob Theodore Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; pspace-hardness of the" warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[2] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[3] Michal Čáp, Jiří Vokřínek, and Alexander Kleiner. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[4] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *2012 IEEE International Conference on Robotics and Automation*, pages 3293–3298. IEEE, May 2012.

[5] Daniel Martin Kornhauser, Gary L Miller, and Paul G Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. Master's thesis, M. I. T., Dept. of Electrical Engineering and Computer Science, 1984.

[6] Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. *Algorithmica*, 2(1-4):477–521, 1987.

[7] Michal Čáp, Peter Novák, Alexander Kleiner, and Martin Seleckỳ. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE Transactions on Automation Science and Engineering*, 12(3):835–849, 2015.

[8] David Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.

[9] Jingjin Yu and Steven M LaValle. Optimal multi-robot path planning on graphs: Complete algorithms and effective heuristics. *arXiv preprint arXiv:1507.03290*, 2015.

[10] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

[11] Yash Mulgaonkar, Brandon Araki, Je-sung Koh, Luis Guerrero-Bonilla, Daniel M Aukes, Anurag Makineni, Michael T Tolley, Daniela Rus, Robert J Wood, and Vijay Kumar. The flying monkey: A mesoscale robot that can run, fly, and grasp. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4672–4679. IEEE, 2016.

[12] Richard J. Bachmann, Frank J. Boria, Ravi Vaidyanathan, Peter G. Ifju, and Roger D. Quinn. A biologically inspired micro-vehicle capable of aerial and terrestrial locomotion. *Mechanism and Machine Theory*, 44(3):513–526, 2009.

[13] Daler Ludovic, Mintchev Stefano, Stefanini Cesare, and Floreano Dario. A bioinspired multi-modal flying and walking robot. *Bioinspiration & Biomimetics*, 10(1):016005, 2015.

[14] A. Kalantari and M. Spenko. Modeling and performance assessment of the hytaq, a hybrid terrestrial/aerial quadrotor. *Robotics, IEEE Transactions on*, 30(5):1278–1285, 2014.

[15] Jared R Page and Paul EI Pounds. The quadroller: Modeling of a uav/ugv hybrid quadrotor. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4834–4841. IEEE, 2014.

[16] SH Jeong and S Jung. A quad-rotor system for driving and flying missions by tilting mechanism of rotors: From design to control. *Mechatronics*, 24(8):1178–1188, 2014.

[17] Syma X9 Flying Car. http://www.symax9.com/.

[18] Alex Kushleyev, Daniel Mellinger, and Vijay Kumar. Towards a swarm of agile micro quadrotors. In *in Robotics: Science and Systems (RSS*, 2012.

[19] M. Turpin, N. Michael, and V. Kumar. Decentralized formation control with variable shapes for aerial robots. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 23–30, May 2012.

[20] Y. Mulgaonkar, G. Cross, and V. Kumar. Design of small, safe and robust quadrotor swarms. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2208–2215, May 2015.

[21] Wolfgang Hönig, Christina Milanes, Lisa Scaria, Thai Phan, Mark Bolas, and Nora Ayanian. Mixed reality for robotics. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 5382–5387. IEEE, 2015.

[22] Benoit Landry. *Planning and control for quadrotor flight through cluttered environments*. PhD thesis, Massachusetts Institute of Technology, 2015.

[23] Hao Jiang, Morgan T Pope, Matthew A Estrada, Bobby Edwards, Mark Cuson, Elliot W Hawkes, and Mark R Cutkosky. Perching failure detection and recovery with onboard sensing. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1264–1270. IEEE, 2015.

[24] Mike Phillips and Maxim Likhachev. Sipp: Safe interval path planning for dynamic environments. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5628–5635. IEEE, 2011.