Localization and Unpredictable Surveillance with Drones

Rafael Calleja, Sukrit Arora, Tiffany Cappellari
University of California, Berkeley

***Abstract*— This design project aims to implement Daniel Fremont's Reactive Control Algorithm for Unpredictable Surveillance on an improviser drone that satisfies specific goals and safety constraints while exhibiting some level of randomization. For each execution, design goals include: ensuring the drone traverses all predetermined targets, avoiding collision with a patrolling adversary, and exhibiting unpredictability between executions. The algorithm is realized using a Crazyflie drone improviser against either a simulated or Kobuki adversary. Our project code can be found in this GitHub repository.**

I. INTRODUCTION

Our project is Localization and Unpredictable Surveillance with Drones (and Kobukis). The goal of this project is to implement Daniel Fremont's Reactive Control Improvisation (RCI) algorithm beyond simulation. The RCI algorithm attempts to synthesize a "reactive system" which chooses actions for an improviser in response to the environment (some adversary), ensuring the desired specification is satisfied no matter what the environment does.

Our initial goal was to be able to run the algorithm with the Crazyflie drone as the main improviser against a simulated adversary. We later added a Kobuki adversary as a challenge. In this project we attempted to demonstrate understanding of and incorporate Finite State Machines (FSMs), Wireless Communications (radio and WiFi), Linear Temporal Logic (LTL), Sensors (infrared localization), and Actuators (Crazyflie drone and Kobuki).

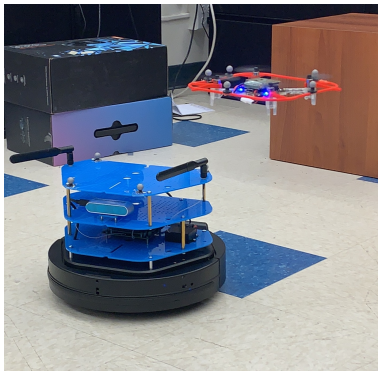

Fig. 1. The Kobuki adversary and the Crazyflie improviser. The grey balls atop each robot are passive IR beacons necessary for tracking.

II. METHODS

Our project included the use of a Crazyflie 2.1 drone, a Kobuki, IR beacons/sensors, and WiFi/radio to implement the discrete Reactive Control Improvisation (RCI) algorithm in continuous space. The Crazyflie 2.1 drone is used as the improviser while the Kobuki is the adversary. The Crazyradio PA was used to communicate with the Crazyflie 2.1 for motion commands and battery life information. WiFi was used to communicate with the Kobuki for motion commands. The motion capture Optitrack setup in Cory 337 is fitted with IR sensors and beacons sending a central computer localization (orientation and position) information.

A. Reactive Control Improvisation (RCI)

The Reactive Control Improvisation [1] algorithm was created by Daniel Fremont. It takes in some specification (i.e. alphabet, targets, hard and soft constraints) and constructs a finite state machine accordingly. The FSM is constructed with accepting and failure states that signal to the algorithm whether the specifications have been satisfied or violated.

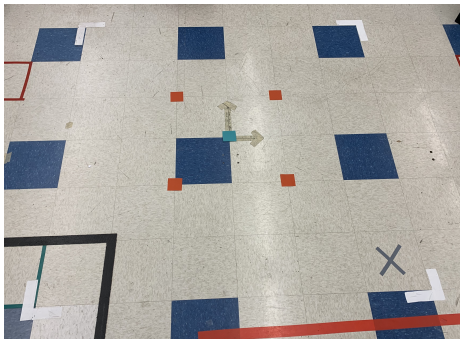

Fig. 2. The 7x7 grid in which the RCI algorithm was implemented. The red markers indicate the waypoints and the white tape indicate the corners of the grid.

Due to a lack of hardware (namely the Crazyflie drone) in the beginning of the project's life cycle, much time was spent learning about Daniel Fremont's algorithm. The algorithm is a finite state machine defined by hard consraints, which must be strictly satisfied, and soft constraints, which may be violated according to some hyperparameter, (both expressed as Linear Temporal Logic expressions) as well as the actions or "alphabet" of an improviser and an adversary.

For this project we used the following constraints:

Hard Constraints:

1) *The drone must visit every waypoint.* This is implemented using a one-hot encoding of whether or not a waypoint has been visited.
2) *The improviser drone must not collide with the adversary.* To monitor this property, we track both the drone's

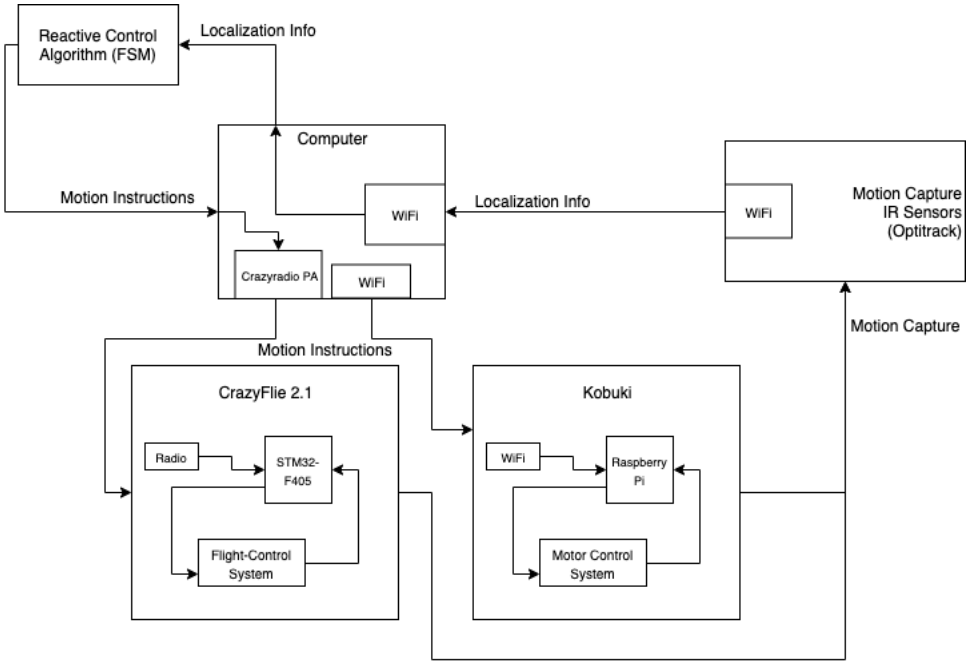

Fig. 3. The project architecture diagram.

position and the Kobuki's position as (x, y) coordinates and ensure that the two are never equal.

3) *The adversary may not enter specific locations: the improviser's target locations as well as the center coordinate.* This is to make the problem solvable. We enforce this property by hard coding the illegal locations into the adversary's path planning.

Soft Constraints: *The drone must visit every location without repeats.*

Alphabet: [Forward, Backward, Left, Right]

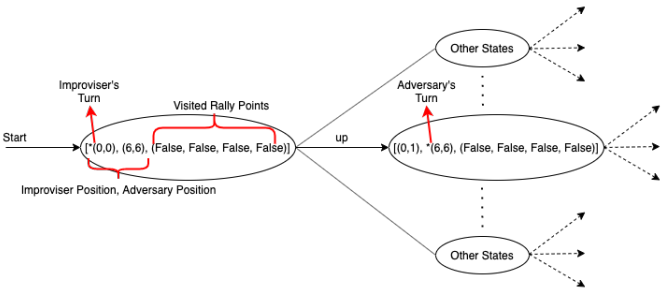

Fig. 4. An example of an FSM generated by the RCI algorithm. Each node of the FSM was an encoding of the state of the algorithm: [(Improviser Position), (Adversary Position), (One-Hot Encoding of State Visitation)]. The collision constraint is encoded as failure states within the state machine.

This finite state machine is traversed according to both the improviser's and adversary's actions, attempting to satisfy the constraints as it does so.

Figure 5 below shows a few simulations of this algorithm at work with the green triangles representing the starting positions of the adversary and the improviser, the dotted line showing the path of the adversary, the solid line showing the path of the improviser, and the red circles representing the way points.

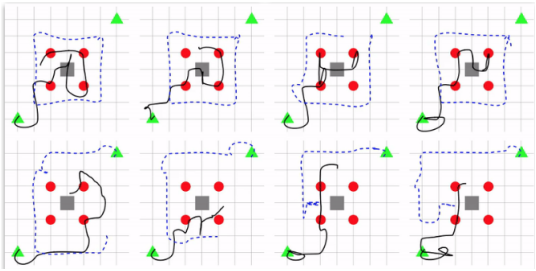

Fig. 5. Simulations from Daniel Fremont's website. The top row of simulations demonstrates a fixed-loop patrolling adversary while the bottom row demonstrates a reactive-chasing adversary. The randomness in the algorithm is apparent in its non-determinism given the movement of the fixed-loop patrolling algorithm [2].

The algorithm as implemented assumes a discrete, 7x7 grid environment in which both the improviser and adversary have 4 possible moves: forward, backward, left, and right. It also assumes that the improviser cannot move into specific grid coordinates, specifically the center as well as the waypoint locations. The drone receives commands from the Reactive Control Improvisation algorithm and the adversary can be switched between 3 different modes: fixed-loop patrolling, reactive-chasing, and human-controlled.

B. Localization

Localization was implemented using the Opitrack Motion Capture system. This localization method works by using a series of ten, high data rate, low latency infrared LED sensors. These sensors continuously emit IR pulses that are reflected by the passive IR beacons fitted on each of our robots. The system then uses the time of flight information along with the over-determined data from the ten IR camera system to resolve a series of linear equations and accurately provide location and orientation information in a three dimensional space.

One problem we faced was the difference in the standard axes as defined by our two interfacing systems. While Optitrack defaulted its axes to [X: Forward, Z: Left, Y: Up], ROS expected axes aligned as [X: Forward, Y: Left, Z: Up]. To resolve the motion capture system with ROS, we had to transform the output data of the motion capture system by reflecting and swapping data axes to align with ROS's expected input axes. We used the `vrpn_ros_client` package for Ros-Optitrack communication. By creating Rigid Bodies in the Optitrack System and pointing the `vrpn` client to the same IP address that Optitrack broadcasted data on, we were able to create ROS topics that received the position and orientation information of the improviser and adversary respectively. We were then able to incorporate this information in our codebase by subscribing to these topics.

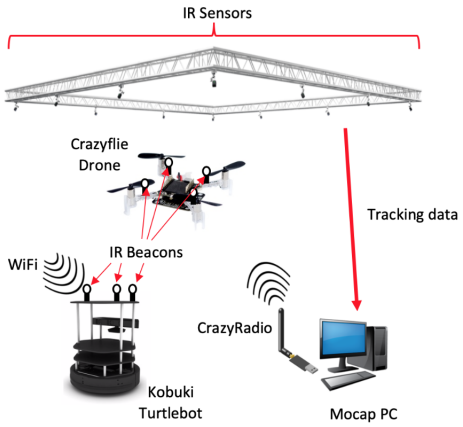

Fig. 6. A diagram of our motion capture room setup. While not shown in the diagram, each component of the setup was connected to each other through ROS topics and services.

C. Robotic Operating System (ROS)

Initially ROS's asynchronicity seemed necessary in order to achieve a real-time execution of the RCI algorithm. In addition, Tiffany already knew ROS as an EECS 106A TA. However, after speaking with Fremont, we realized that the algorithm was inherently turn-based and therefore, asynchronicity in terms of robotic movement was no longer useful. While it was no longer a necessity we eventually settled on implementing our design in the ROS environment as there was existing infrastructure and documentation for it in the motion capture room's Optitrack software.

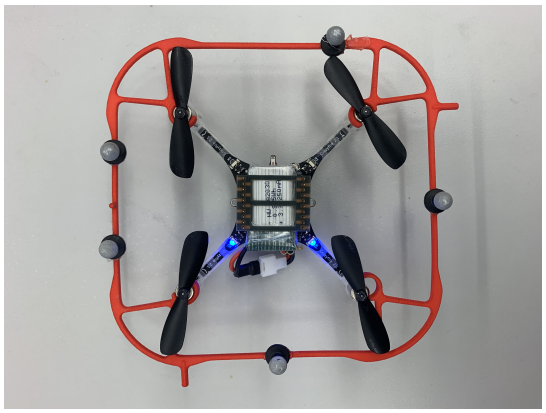

Fig. 7. The Crazyflie drone fitted with a 3D printed frame to mount the IR beacons for localization.

ROS creates a subscriber-publisher relationship between nodes through topics. The publisher will send information to a topic and the subscriber will continuously read the topic and run a callback function whenever information is received through it. For our project, we created ROS nodes for both the Kobuki adversary and the Crazyflie improviser. We set up two topics for them to communicate through (`/adversary` and `/improviser`) to tell each other when its current move was complete. We also created topics for receiving IR sensor position and orientation information (`/cf/pose` and `/adv/pose`) and topics for sending movement commands (`/ref` and (`/mobile_base/commands/velocity`) for each robot. A diagram of this can be seen in figure 8.

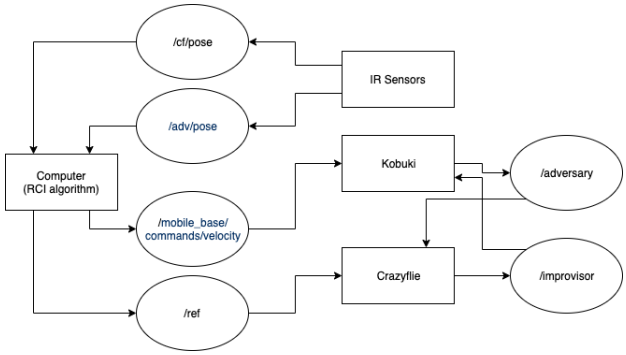

Fig. 8. A diagram of our ROS communication. The ovals are the topics and the rectangles are the nodes.

The actual communication of data depended on the device. For movement instructions for the drone, the LQR controller would output the goal angle and thrust information that would go directly into data packets that were sent to the drone over the Crazyradio. The rest of the communication happened over the LAN of the WiFi network by setting static IPs and adjusting settings in ROS.

Additionally, while the Crazyflie drone came with a built-in LQR controller, we had to create our own custom proportional controller for the Kobuki in order to drive it to the correct position and orientation. In order to do this we

took the goal's position in the world frame and transformed it into the Kobuki's frame and then leveraged the Kobuki's rotational and transnational motion commands to calculate the linear and angular velocities needed to move it to the specific location.

It was also necessary to reconfigure the network communications because ROS creates a ROS Master Node that connects all the topics and allows nodes to communicate through them. Initially, the computer and the Kobuki both had their own Master Nodes, making communication between them impossible. While the simplest solution would have been to point the computer's Master Node to the Kobuki's, this would mean the Raspberry Pi on the Kobuki would have to function as the central computer and run the algorithm but it did not have the necessary computational power. We instead chose to point the Kobuki's Master Node to the computer by establishing namespaces for both the Raspberry Pi and central computer ROS environments, and as mentioned before we resolving the networking issues by setting a static IP address on the central computer.

D. Integration

This project had a lot of individual self-contained pieces that needed to work together for the final result. While there were several aspects of integration that were time consuming, such as the ROS-Optitrack and ROS-Kobuki communication, we'd like to focus less on the implementation details of integration and focus more on the high level changes that needed to be made to integrate the parts of the project.

Specifically, because the RCI algorithm operates in a discrete grid space, and the Optitrack and ROS systems provide localization information and control commands in a continuous coordinate system, we had to design a way to translate information back and forth between these domains to make the end-to-end system work. By taking several distance measurements, we created a mapping between these two domains. The mapping is subjective, as many continuous points map to a single grid point. However, by intelligently picking when we convert, we were able to integrate the many moving parts of our project without losing too much accuracy. The algorithm is also executed in a discrete-time, turn-based fashion (i.e. first the improviser moves, then the adversary moves, etc.). In order to ensure that this turn-based ordering was never violated we took advantage of ROS's topics to publish signals when a move started and completed. This, in combination with a hierarchical state machine to ensure that each robot idled or moved depending on the state of the other, allowed for variable-length move times critical to the algorithm's operation.

III. RESULTS

We were able to implement the RCI algorithm in the real world with the Crazyflie drone as an improviser and a Kobuki as the adversary. We successfully implemented three different adversary strategies: fixed-loop patrolling, reactive-chasing, and human-controlled. You can see the fixed-loop and chase implementations in action at this YouTube link.

While the algorithm executes as expected, there are several limitations that arise from the non-ideal conditions of the real world. The first is that the accuracy of the improvise deteriorates over time due to the extremely limited battery life of the Crazyflie. As the battery voltage drops, the drone grows more unstable until it is no longer able to sustain the command given to it by the LQR controller and it crashes. Another issue is finding the optimal state space size. A larger state space translates to a finer grid, which would lead to smoother and seemingly faster movement (as each decision is made at a grid point, and transitions to grid points would take less time if grid points were closer together). However, with a larger state space we encounter both time and memory issues as the offline FSM generation would take too long to generate and be too big to store. Additionally, sometimes the Kobuki adversary would not be quite on its mark after moving to a new position. This is due to a combination of a few things: (1) our closed loop controller on the Kobuki was not as robust as the Crazyflie's built-in LQR controller but it would attempt to correct itself in its next movement, (2) the Kobuki was much larger than the drone and physically took up more space, and (3) the center of the Kobuki to the IR sensors' was not the center of the actual Kobuki itself because of the placement of the IR beacons.

IV. CONCLUSION

This project has been a great opportunity to experience, firsthand, the pitfalls of (1) translating something purely discrete into a continuous space, (2) integrating many disjoint components into a single streamline system, and (3) working with unfamiliar technologies and exercising professional communication.

Initially the goal of this project was to integrate Daniel Fremont's Reactive Control Improvisation algorithm onto a localized Crazyflie drone improviser such that it can react to a simulated adversary and satisfy specific hard and soft constraints. This goal evolved to include a Kobuki adversary. Through simulation, we were able to show that an improviser could satisfy all its constraints against a simulated adversary. We proceeded to physically realize this by setting up localization for our drone and Kobuki using the Optitrack Motion Capture system and ROS environment. We set up radio communication to the Crazyflie and later WiFi communication to the Kobuki through the subscription and publishing of ROS topics. A section in the motion capture room was sequestered into a 7x7 discrete grid in order to comply with the algorithms expected environment.

While the Crazyflie drone was well documented with an existing LQR controller implementation readily available, the Kobuki was less equipped to integrate into the Opitrack-ROS system. Given the drone's many degrees of freedom, moving to an approximate grid coordinate was simple, however it was necessary to implement a closed-loop, proportional controller on the Kobuki to counteract any compounding drift.

Future work for this project include modifying the Reactive Control Improvisation algorithm to account for uncertainty in robot movement or enabling the robots to run

in continuous time/space. Upon discussing such prospects with Daniel Fremont, a couple approaches were theorized, but were ultimately out of scope of both EECS 149 and the time frame of this final project. One such approach to implementing uncertainty is feeding the improviser a command from the RCI algorithm and observing its actual movement as an input back into the FSM. A different approach to implementing the execution in ”continuous” time/space was to increase the resolution of the discrete grid from 7x7 and adding RCI and externally defined constraints on the possible failure states (i.e. distance from improviser to adversary must be greater than some threshold according to the dimensions of each entity).

Hypothetically, as the title of the project suggests, the development and study of this algorithm and its implementation in the real world may prove useful in unpredictable robotic planning. If the algorithm proves to be solvable even without its current assumption of perfect information (of both the improviser and its environment), this project may be a step towards online verifiability for robotic systems.

ACKNOWLEDGMENT

This research was supported by the EECS Department, University of California, Berkeley and the Sastry Group. We thank Professor Sanjit Seshia, Professor Prabal Dutta, GSIs Edward Kim, Jean-Luc Watson, and Neal Jackson from the EECS Department, University of California, Berkeley as well as Professor Shankar Sastry for letting us use his lab's motion capture room, his student Valmik Prabhu from the ME Department, and Daniel Fremont who provided insights and expertise that greatly assisted the project.

REFERENCES

[1] D. J. Fremont and S. A. Seshia, “Reactive control improvisation,” 2018.
[2] D. Fremont, “Reactive control improvisation.”