iRobot Pursuit:
Hybrid Simulation using Accessors in Ptolemy II

Nikunj Bajaj, Shromona Ghosh, Marten Lohstroh
Final Report EECS 249A Project, Fall 2014

December 19, 2014

1 Introduction

In this project, we aim to design, model and analyze a cat-and-mouse game[1] between robots. The game finds its application in automated security, context-aware computing, and the Internet of Things. A key challenge in these types of applications and their dynamically evolving infrastructure is that the distinction between "design time" and "run time" becomes blurred. Ensuring that different components and subsystems can be dynamically recombined yet still function properly requires highly advanced development methodologies, models, and tools [3]. We will leverage one of these, namely *accessors* [5], which allow simulated components to be replaced by their real-world counterparts in composition with other components that continue to be simulated.

The result of integrating simulations with deployed cyber-physical systems is called (real-time) *hybrid simulation* [2] [2], a strategy used in structural and civil engineering that allows a critical component—in our case the iRobot—to be isolated and physically tested while still capturing the dynamic behavior of its interaction with the entire system. We expect techniques like these to be become increasingly important in the field of modeling and simulation of cyber-physical systems at large.

2 Methodology

More than just implementing a game using real robots, our goal is to study the behavior of its interacting components; an intruder and several guards that protect an asset. For instance, we are interested in whether the intruder can reach the asset, or more importantly, reach it and escape without being caught. Because any assertion about the possible behaviors in the game is based on a *model*, we would like to test the assumptions that gave rise to that model by observing the physical interactions between the robots and their environment, and if possible, refine that model. In this project we started exploring the latter by building a control system in Ptolemy II that interacts with iRobots through accessors. Subsequently, formal verification of the properties of our system could be done, for example by compiling our Ptolemy II model into a SpaceEx [1] specification, but this lies outside of the scope of this project.

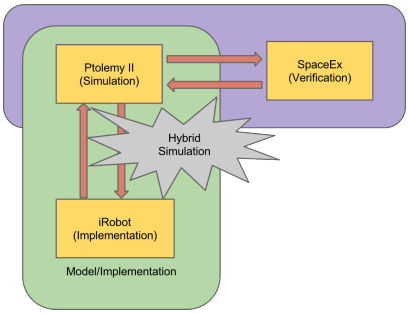

Figure 1: Hybrid simulation and formal verification.

Accessors mediate interactions with arbitrary subsystems through the exposure of an actor interface. An accessor defines inputs, outputs, and a limited set of methods that perform some computation and coordinate the exchange of data with external resources when the actor is "fired". Abstract actor semantics allow actors and accessors to be composed, and under governance of a particular model of computation a composition acquires its execution semantics. Because accessors are still in early development, only a limited number protocols are supported by the host, Ptolemy II. Support for RESTful interactions is currently most mature, so we decided to use HTTP to interact with to the iRobot.

2.1 System Architecture

2.1.1 Hardware

We used the iRobot Create to which we communicate over UART using a BeagleBone Black, connected via a SparkFun BOB-12009 bidirectional logic level converter. The BeagleBone is powered by the iRobot by means of a SMAKN DC/DC step-down power converter. Furthermore, through I2C we connected a LSM303 Triple-axis Accelerometer + Magnetometer, and through USB we connected a TP-LINK TL-WN725N WiFi and an Asus BT400 Bluetooth module.

2.1.2 Software

On the BeagleBone we used a regular ARM Linux kernel. We implemented a web server in Python using the

[1] A contrived action involving constant pursuit, near captures, and repeated escapes.

[2] Not to be confused with the simulation of hybrid systems, in which continuous behavior is specified by differential equations along with discontinuous changes specified by discrete event switching logic.

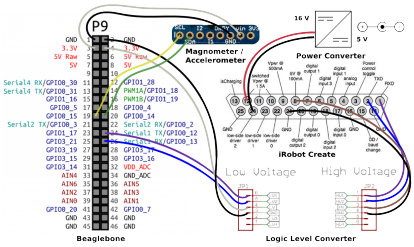

Figure 2: Diagram of the hardware components.

GEvent library and its Webserver Gateway Interface library. Through a library called Pyrobot, written by Damon Kohler, we actuate the iRobot and read its sensors. We use polling to retrieve sensor readings.

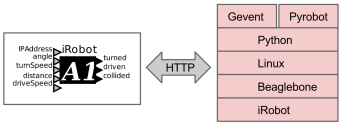

Figure 3: Software stack.

The interface exposed by the web server is simple; arguments of an HTTP GET request specify the angular and linear displacement of the requested maneuver. For each request, the robot first turns and then drives straight. A turn is only taken if the provided angle is not zero and the turn speed is greater than zero. The robot only drives if a non-zero distance is provided and the drive speed is greater than zero. Negative angles result in clockwise motion, positive angles make the robot turn counter-clockwise. Similarly, negative distances make the robot drive backwards and positive distances result in forward movement. The server blocks until the maneuver is completed. An example request looks like this: `http://robot1:8088/?turnSpeed=200&angle=45&driveSpeed=200&distance=200`. Angles are measured in degrees, distance is measured in millimeters, and speeds are measured in mm/second.

2.1.3 Model

The system was modeled in Ptolemy II, where a pursuit game was simulated with two chasers and one intruder in a virtual environment consisting of a square of size 400×400 cm. The scenario is depicted in Figure 4, the Ptolemy model is shown in Figure 5.

As tested empirically, and reported in section 3, we are unable to use our range (Bluetooth RSSI) and orientation (magnetometer) sensors. Hence, we state the following assumptions:

- Movement of the robots is restricted by predefined step size (controlled by a parameter).
- If the intruder comes within a threshold distance of one/more of guards(s) it is deemed to be caught.
- All robots will have known start positions and their position continues to be estimated based on dead-reckoning.

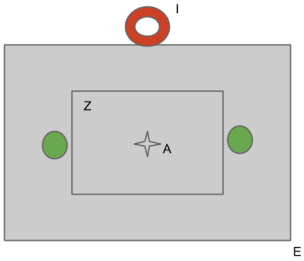

Figure 4: The environment E features some valuable asset A that is protected by guards G. The objective of intruder I is to reach and capture the asset and then escape, evading the guards. The guards will come into action when I enters critical zone Z, and will chase down the intruder.

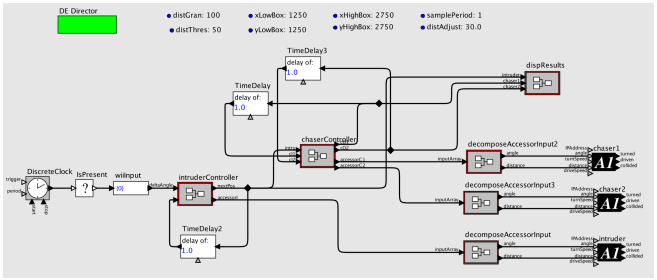

Figure 5: Ptolemy Model.

Key components in the model are the "Intruder Controller" and the "Chaser Controller" composite actors. The intruder controller takes in the angle input given by a WiiMote controlled by a user and calculates the next position of the intruder based on the current orientation and location. The chaser controller takes as input the location and orientation of the chasers and intruder and estimates the next best location for the chasers. In doing this, the controller implements two preliminary strategies:

- It makes the chasers move toward the last reported coordinate of the intruder.
- It makes the chasers move towards the next predicted location of the intruder based on its posit on and orientation.

The guards operate in two modes namely *idle* and *chasing*. The chasing mode gets activated when the intruder enters the "critical zone" of the environment.

Because the `httpRequest()` calls in the iRobot accessor are blocking, our simulation features a strictly interleaved operation of the robots. We could potentially make the Web server on the BeagleBone non-blocking, but then we would have to allow sufficient time between request, or we may attempt to actuate the robot while it is still handling a previous request. Alternatively, we could run a Web server in Ptolemy II to receive a callback upon completion of a maneuver. Since an interleaved execution would suffice to demonstrate a proof-of-concept, and we favor determinism over responsiveness, no asynchrony is featured.

3 Results

3.1 Actuation of the iRobot

3.1.1 Angle actuation errors

Angle actuation errors occur due to several factors such as speed of the robot, friction in the wheels and the polling interval. We orthogonalize these issues and setup experiments to solve each of them separately. We first conduct an experiment to find the optimum turn speed, and then perform an experiment to find an error model for the angles.

To find the optimum turn speed we vary the turn speed and measure the error. We repeat the experiments for turn speed varying from 50 mm/s to 300 mm/s in increments of 25 mm/s and different turn angles, namely 30°, 60° and 360°. The following graph summarizes the results. The

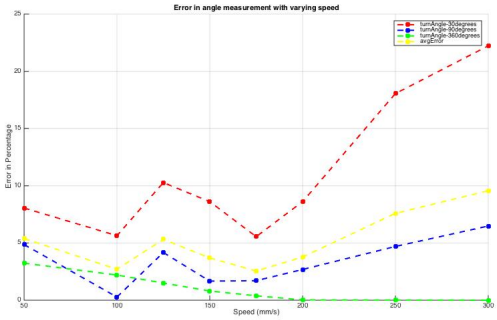

Figure 6: Error in angle measurement with varying speed.

graph shows four traces. The red trace shows the normalized error for different turn speeds for a turn angle of 30°. The blue trace is for a turn angle of 60° while the green trace is for a turn angle of 360°. The yellow trace shows the average normalized error of the red, yellow and blue trace.

We consider only those speeds which yield an average normalized error of less than 5%. In the graph, we see the speeds that satisfy this criterion are 100mm/s, 150mm/s, 175mm/s and 200mm/s. We choose 200mm/s to be the optimum turn speed as it gives us a good trade off between responsiveness and error.

We now repeat the experiments by varying the required turn angle for a turn speed of 200 mm/s. The following graph summarizes the normalized error for different turn angles varying from 15° to 360° in increments of 15°.

The normalized error has an inverse relation with the turn angle. To correct this error, we took two approaches:

1. We experimentally found out the correction required for every range of turn angles.

2. We tried to fit a curve to our experimental data. We tried to fit 2 curves, an exponentially decaying and an hyperbolic curve.

The following graph shows the two fitting with the actual data. The error from the curve fitting is about 0.2694 for exponential fitting and 0.0878 for hyperbolic fitting.

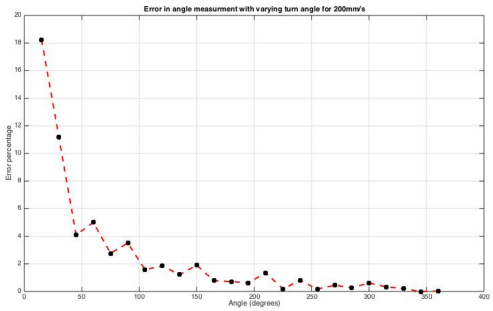

Figure 7: Error in angle measurement with varying turn angle for 200mm/s.

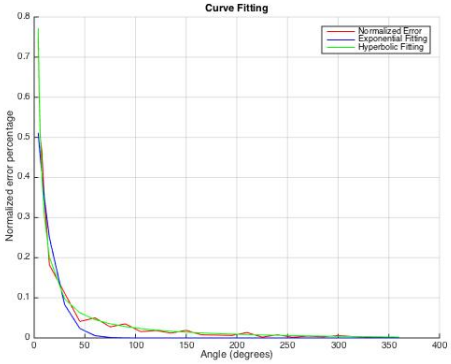

Figure 8: Curve Fitting

We compared the results of the curve fitting to the corrections we found experimentally, and saw that the results were comparable. The hyperbolic fitting has less deviation from the experimental results and is less expensive to implement compared to the exponential fitting. Hence, we go for the hyperbolic fitting for correcting our angle actuation errors. The corrected angle θ in terms of the required angle α is :

$$\theta = \alpha(1-y) \quad where \quad \alpha = x \tag{1}$$

3.1.2 Distance actuation errors

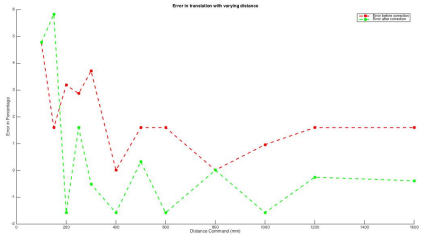

Figure 9: Distance actuation errors.

We varied the distance to be traveled by the robot from 100mm to 1600mm at a speed of 200 mm/s. We found that the normalized error falls into a range of $[0, 0.05]$. For simplicity, due to the closeness of values, we do a linear fitting. The correction factor is the average of the normalized errors. We see the average is 0.0196. The graph

shows the normalized errors, before and after applying the correction.

From the graph we notice that the correction improves errors only for required distances greater than 200mm. Thus, we perform no correction for required distances less than 200mm. The correction for distances greater than 200mm is linear model where $y = 0.0196 * x$.

3.2 Bluetooth received signal strength indication

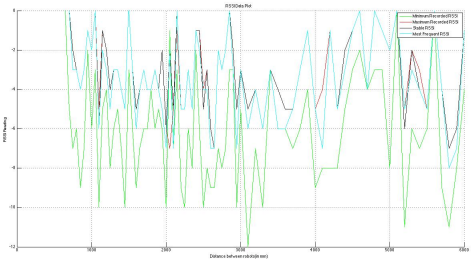

Figure 10: RSSI Data Plot.

The RSSI values were highly erratic and failed to give us a proper model for the distance to RSSI relation. This could have been for a number of reasons such as reflection and interference. Also RSSI works well for distances over one meter in heavily controlled environments such as an anechoic chamber [4], but our environment was anything but controlled, and we were in fact trying to find a suitable model for smaller distances with a least count of 100mm.

3.3 Magnetometer

We tried to model the magnetometer. However, due the presence of iron beams in the floor of our environment, we were unable to get any meaningful results from the magnetometer.

3.4 Ded-reckoning

Ded-reckoning is typically not ideal for location estimation because of the accumulation of actuation error. However, because of the strategies pointed out in Section 3.1.1 and the limited size of the environment, we could implement the pursuit game with reasonable accuracy. We tried to test the limit of ded-reckoning. We gave a robot 50 instructions and checked the final position of the robot. We repeated this experiment 10 times and recorded the following:

$$Expected \quad end \quad location : [196.6, 328.3]$$

$$Range \quad of \quad x : (10.75, 204.47)$$

$$Range \quad of \quad y : (304.75, 319.99)$$

$$Average : [174.05, 313, 767]$$

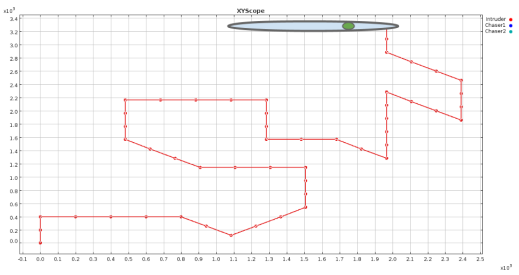

Figure 11: Ded-reckoning experiment.

4 Conclusion

In this project we explored the methodologies and challenges in implementation of a intruder pursuit game. We first modeled the actuation errors of the iRobots. We further investigated different ways to obtain the range measurements and orientation of robots. We concluded RSSI values do not correlate well with small distances and need a very controlled environment to not be affected by reflection and interference. On the contrary, ded-reckoning can be brought to acceptable accuracy if the actuation error is taken into account in making control decisions.

Future work could be dedicated to: experimentation with more sophisticated control strategies; achieving higher operating speeds, more asynchrony, and tighter bounds on latency; development of non-blocking accessors and communication to Websockets to facilitate two-way communication; use of other sensors to provide better estimates of position and orientation. Finally, a video presentation of our demo is available here: `http://youtu.be/y0ax3vaWyxY`.

References

[1] Spaceex. `http://spaceex.imag.fr/`.

[2] R. Christenson, Y. Lin, A. Emmons, and B. Bass. Large-scale experimental verification of semiactive control through real-time hybrid simulation. *Journal of Structural Engineering*, 134(4):522–534, 2008.

[3] E. A. L. et al. The terraswarm research center (tsrc) (a white paper). Technical Report UCB/EECS-2012-207, EECS Department, University of California, Berkeley, Nov 2012.

[4] J. Jung, D. Kang, and C. Bae. Distance estimation of smart device using bluetooth. In *ICSNC 2013, The Eighth International Conference on Systems and Networks Communications*, pages 13–18, 2013.

[5] E. Latronico, E. A. Lee, M. Lohstroh, C. Shaver, A. Wasicek, and M. Weber. A vision of swarmlets. *TBD: A Suitable Magazine, Journal, or Conference*, 2014.