Boston University
Electrical & Computer Engineering
EC500 Cyber-Physical Systems

Final Project Report

Physical Masquerade Attack Detection-Implementation of Path Finding and Attack-Proof Path Finding Algorithm

Team Members:
Yuqiang Ning
Ting Wang
Zachary Bachrach
Jiayue (Cindy) Bai

Introduction

Nowadays, more and more autonomous mobile robots are used across the factory floors to increase efficiency and lower human efforts. However, their rising popularity also raise security concerns. In short, autonomous unit lacks an effective monitoring method. The term "masquerade attack" usually refers to soft attacks that uses fake identities, such as a network identity, to gain unauthorized access.

In this project, we looked into physical masquerade attacks, where a compromised robot can disguise as a good robot and gain access to restricted zones. In the context of attack proof multi-agent path finding (APMAPF) problem, it is about a robot deviating from its predefined path and reaches a restricted "safe zone" without being noticed.

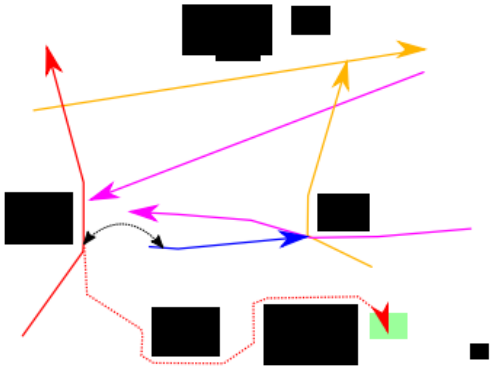

Fig 1:a compromised robot reaching "safe zone"
Source: Wardega, Tron, Li 2019

In this project, we studied a potential solution to prevent physical masquerade attacks with an attack-proof pathing finding algorithm proposed by Kacper etc[1], implemented it on a two agent system consisting of one iRobot and one fixed Pixy2 camera. In addition, we also realized the dynamic path planning function, which allows the iRobot to detect unknown obstacles, recalculate and execute new path automatically.

Design Diagram

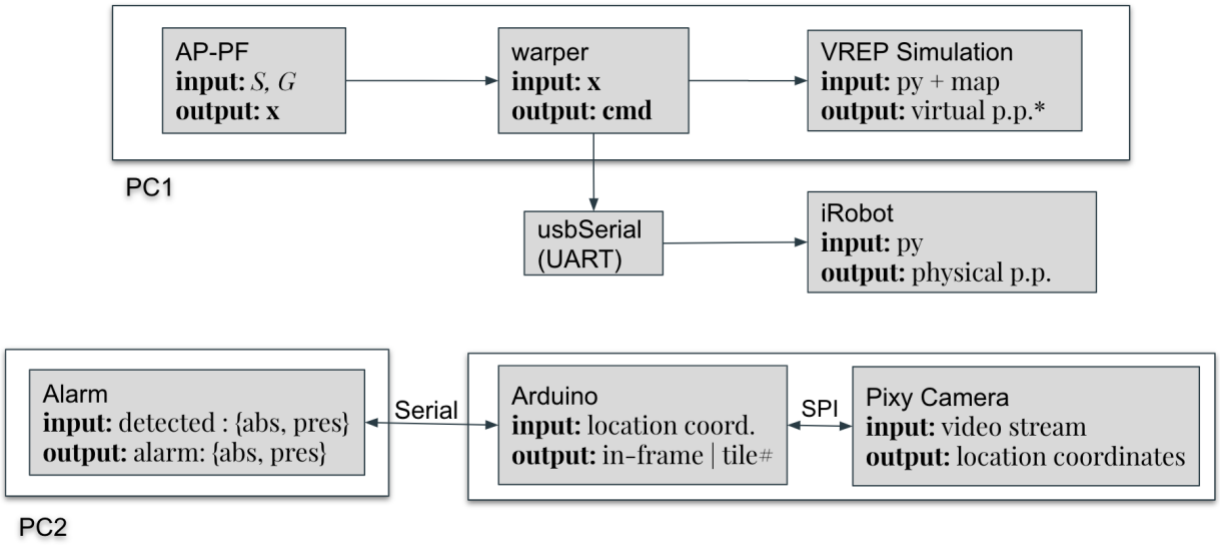

Fig 2: design diagram

Note:

- p.p. : posture and position
- Camera is a fixed "robot"
- S for start state set
- G for goal state set
- **x** for a sequence of x, the semantic of which is a feasible attack proof path, or a sequence of states.

Specification and Modeling

- Attack-Proof Path Finding Algorithm
  - With a map as input, generate an attack-proof path to get from starting to end point while passing through the camera monitor region.
  - When unknown obstacles detected, generate a new attack-proof path with current position as starting point.
- Alarm Generation:
  - If attack happens, trigger alarm.
  - If iRobot is detected before *timewindow1*, trigger alarm.
  - If iRobot is not detected within *timewindow1* and *timewindow2*, trigger alarm.
  - If iRobot is detected after *timewindow2*, trigger alarm.

Software/Hardware Implementation

Path Finding and attack proof path finding Algorithm

The logistic of the path finding and attack proof path finding algorithm is to describe and encode the problem using CNF formulas, and feed them into solvers like msat, picosat etc to solve them. Specific rules for encoding these CNF formulas can be found in Kacper etc's paper [1]. The Python coded CNF rules are then fed into solvers using a public library "pySMT" [2], which could automatically transform our rules into solver specific language.

File for above process in our project is __gridworld_safezone_efsmt.py. Which is an adaptation based on Kacper's code. We modified it to fit the scenario of a fixed monitoring camera.

Obstacle detection and reaction

The bumper of iRobot create2 will return *true* when bumping into obstacles and *false* otherwise. So in our code main.py, if the signal changes from false to true when iRobot is running, the code will stop there, break up current commands, store current position & obstacle position, return to the start of the program and recalculate the path by calling apmapf algorithm again using new starting point and obstacle information. Every time there's a bump, the program will store the obstacle location in a list, meaning that our program could memorize obstacles it meets, and thus will not repeat previous bumps again.

Changing coordinates into iRobot movements

Lacking sufficient hardware, it's hard to implement accurate localization in our iRobot. Instead it only knows where it is by putting it in a coordinate known starting point and remembering its movement steps.

Also the lack of localization makes it impossible to know its direction. To tackle this, we gave it an original direction by manually putting it that way and generates future directions according to the steps it will move. For example, when the robot moves from (1,2) to (1,3), then the next step direction would be (1-1 , 3-2) = (0,1).

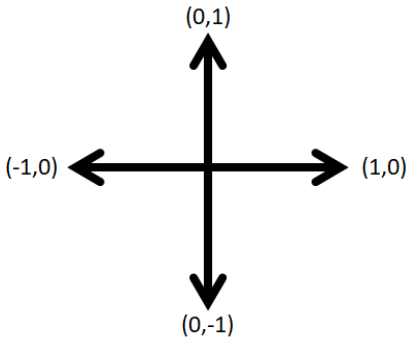

Fig 3: directions

The rule of turning for the iRobot to execute is shown below in figure 4, in which column 1 is its current direction and row 1 is its next step direction.

	(0,1)	**(0,-1)**	**(1,0)**	**(-1,0)**
(0,1)	0°	180°	90°	270°
(0,-1)	180°	0°	270°	90°
(1,0)	270°	90°	0°	180°
(-1,0)	90°	270°	180°	0°

Fig 4: turning rules

We also tuned the step size of our iRobot to be one square of the carpet outside our classroom. So that in every step, it will choose how much degrees to turn and then move one step size after turning.

Code for turning and moving straight one step size is modified based on a public API "breezycreate2" [3]. Logistic of knowing where it is, what the direction is and how many degrees to turn is encoded in main.py.

Robotarium Simulation

Given a family of path of a group of robots, we can verify as well as demonstrate the family of path in Robotarium[4]. The Robotarium project provides a swarm robots remotely for algorithm early test. Both MATLAB and python API are supported. Although MATLAB is preferred as the document said, we select python to implement our algorithm test to maintain better coherence among modules of our project. The algorithm generated family of path is set inside the python script as a numpy array. Note that a path is nothing but a sequence of coordinates. The robotarium map is divided into 3 regions and only the middle region is used in our program for simplicity. The middle region is further divided into 16*16 grid map. The dynamic of the robot can be calculated by programmers from scratch or can be calculated simply by calling API functions to do the same jobs, for example, "si_position_controller(..)".

Robotarium Implementation

Implementation is based on robotarium. Note that once the control code is valid in the simulator, it will work on the robotarium platform, too. Debugging remotely can be tricky, so it might be better to get everything done in robotarium simulator before uploading the code to the platform. Our simulation takes about 15 minutes to finish. Note that the first 34 seconds are set to be initialization, see "robotarium.Robotarium(..)". However, the mini robot has no actual sensor for bumping.

iRobot Simulation

Simulation is based on v-rep. Control of iRobot Model in Vrep simulation environment is based on a public API “pyCreate2” [5]. To do the simulation, we generated the path, changing the coordinates steps into iRobot movement commands using logistic mentioned above, and control it using functions based on pyCreate2.

iRobot Implementation

Implementation is based on iRobot Roomba Create2 Open Interface [6] and python. We use python because we want to keep the project "pure python". However, interrupt mechanism of python is infamous, so our code is in a polling fashion and it actually works. Note that the obstacles in the map are well-defined and we know that every obstacle fits in the little square in grid map.

Monitoring with Fixed Camera

To detect iRobot, there are several ways. Since the map is well structured, using machine learning technique will be a waste. Traditional template matching algorithm will work really well. However, the algorithm is computational intensive compared to a look-up algorithm, which is used by Pixy2. Using Pixy2 will introduce another problem: it is too sensitive to the light condition. So we use several signatures to identify the same object namely iRobot under different light, for example, under shadow.

We have a Pixy2 [7] camera connected to an Arduino Uno microcontroller using the ribbon cable supplied by the Pixy Cam based on SPI protocol. During the demonstration, the Pixy Cam is then taped to the corner of a table, and the Arduino is connected to PC2 serially using the microUSB cable based on UART protocol.

We first uses PixyMon, which is the configuration utility software for Pixy camera, to teach the camera to recognize the robot. Pixy cameras mainly uses color to distinguish objects. Since, the iRobot is green, it is relatively easy to teach the camera to recognize the robot.

After the object is taught to the Pixy Cam, we will be able to use the Arduino Pixy library functions to get access to all different kinds of information of the object once it is detected in the camera frame.

Due to lack of wireless communication capability of the Arduino we are using, we had to pre-define the path of the robot inside the code for alarm generation, so we only need to directly output the x-y coordinates serially to the PC to generate alarms. But we did write a section of code(commented out in the code) that allows the Arduino to output the exact tile number that the robot is on instead of simply x-y coordinates in the frame. So, if we were able to communicate wirelessly between the iRobot and the Arduino, we will then not need to hard code the path inside the alarm generation function; the Arduino will be able to auto-generate the sequence of tiles the robot should pass according to the path generated from the robot, and raise alarm accordingly.

Video Demo: https://youtu.be/tNN2Zmes4x8

Timing/Performance Analysis

At the end, we are able to meet all specifications listed above. We are able to implement the attack-proof path finding algorithm onto the iRobot. The iRobot is able to follow through the path generated by the algorithm correctly. When an unknown obstacle is detected, the algorithm will be notified and generates a new attack-proof path. The only difficulties that we have encountered with the iRobot is that, since the iRobot have to physically bump into obstacles in order to detect them, that physical bump sometimes cause minor displacement in the posture of the robot. As these small displacement cumulates, this will make the robot stop at an inaccurate position since it thinks that it had already reached the goal. This problem is discovered at the last stage of the project, so we did not have the time to actually solve it, but a potential solution to this problem is to change the way we detect obstacles. Instead of physically bumping into it, we can use something like an IR sensor or other kind of rangefinders so that it can detect obstacles at a distance.

The camera component of the project also behave as what we have expected. The Pixy2 Cam is designed to work easily with an Arduino with its own Pixy Arduino library. The camera itself has a frame rate of 60 frames per second, and the camera sends 1Mbits per second to the Arduino. The iRobot is first taught as an object to the camera through PixyMon. Then, the

Arduino will read the data sent from the camera and extract the x-y coordinates of the object in view. We then have the Arduino connected to a PC (PC2) where a python script for alarm generation was running. The script reads the serial output from the Arduino and checks whether an alarm should be generated according to the two time windows specified.

Technical Challenges

- Control of iRobot
    - lack of absolute localization sensors
    - sensor is low resolution and not very accurate
- Cumulation of displacement in posture of iRobot during collision with obstacles
- Lack of remote communication capability of Arduino used

Work Distribution

- *Yuqiang Ning*:
    - Coding for multi-agent path finding and attack proof multi-agent path finding algorithms
    - Coding for obstacle detection and reaction
    - Coding for turning steps (coordinates) into iRobot control commands
    - Demo and reports
- *Ting Wang*:
    - Hardware
    - Alarm generation
    - Coding for obstacle detection
    - Coding for iRobot control commands
    - Demo and reports
- *Zachary Bachrach*:
    - Coding for iRobot control commands
    - Coding for mapf and amapf algorithms, path regeneration
    - Demo and reports
- *Jiayue (Cindy) Bai*
    - Worked with Pixy2 Cam (PixyMon) and Arduino
    - Code for Fixed-Camera Monitoring & Detection
    - Wrote an extra script that allows arduino to output location by tile number or by x-y coordinates
    - Alarm generation
    - Demo and reports

Conclusion

In the end, we successfully implemented an attack-proof path finding algorithm on an iRobot, and were able to monitor it and sends out an alarm when abnormal activities are detected by the fixed Pixy2 camera. In addition, we were also able to achieve dynamic path planning with unknown obstacles so that a new attack-proof path will be generated in reaction to bumping into obstacles on its original path.

Reference

1. Wardega, K., Tron, R. and Li, W., 2019, May. Resilience of Multi-robot Systems to Physical Masquerade Attacks. In 2019 IEEE Security and Privacy Workshops (SPW) (pp. 120-125). IEEE.
2. https://github.com/pysmt/pysmt
3. https://github.com/simondlevy/BreezyCreate2
4. Pickem, D.; Glotfelter, P.; Wang, L.; Mote, M.; Ames, A.; Feron, E. & Egerstedt, M., "The Robotarium: A remotely accessible swarm robotics research testbed" 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore
5. https://github.com/USC-ACTLab/pyCreate2
6. https://www.irobotweb.com/~/media/MainSite/PDFs/About/STEM/Create/iRobot_Roomba_600_Open_Interface_Spec.pdf
7. https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:full_api