AuTom and Jerry: Autonomously Detect and Approach Manually Controlled Mobile Agents

Zhiqi Chen, Sean Huang, Alex Manohar, Tony Pan

December 11, 2019

***Abstract*—There is a cockroach infestation problem in the Mary Markley dorm at the University of Michigan. While chemical sprays can kill some cockroaches, they are harmful for the environment and the health of the residents. A naive solution to reduce roach population is to catch them by hand when one is spotted by a human resident. However, that is not a scalable process unless the procedure can be automated. We are proposing a mobile intelligent robot system that can detect and catch bugs. Our prototype for said solution, named the MBot, will be equipped with LiDAR and cameras to detect the position of a cockroach, chase it into a corner, and catch it for disposal. This solution will ideally reduce the amount of insects prowling the residence halls and allow the university to solve the cockroach-abundance issue, thus removing the stigma from the Mary Markley dorm.**

I. Introduction

Maintaining household hygiene is very important in creating a comfortable living environment for humans. A crucial component contributing to hygiene is the control of pests, specifically the likes of insects such as cockroaches. However, these pests are often hidden and elude the eyes of men, and chasing individually after them as they appear poses a great expenditure in time and effort. A potential solution to this problem is an autonomous agent that can survey the house at all times and swiftly respond to the appearance of these pests. When correctly implemented, the robot can greatly improve both the quality of life as well as the cleanliness of the environment it is deployed in.

In the "Push-Button Kitty" episode of Tom and Jerry, the cat, Tom, is replaced with a robot mouse catcher called "Mechano." The Mechano can find Jerry, the mouse, chase after it, and ultimately capture it. We are proposing to build an autonomous agent similar to the robot cat shown in this video clip: https://www.youtube.com/watch?v=guo-HNbOSwk.

As insect behavior is difficult to model, instead of formulating a set of capturing strategies for arresting the bug, we focused our project on designing a robust finite state machine (FSM) and a fine-tuned controller to detect and pursue the bug efficiently. To test the performance of our robot, we used remote-controlled vehicles to simulate the bug we are chasing. Our project will be presented during the Tishman Hall Expo taking place on December 11, 2019. The demonstration of this robot will not only be interactive but also visually impressive as we will show the real-time video stream and SLAM map.

One important technology for autonomous robotics is using computer vision to acquire, process, and interpret the sensory information of the environment gathered specifically from camera feedback. At the lower level, the sensory information can be used for the control laws commanding the robot motion. At higher levels, it can be used to create models for manipulators to interact with the environment [1]. After the detection of an object, pose estimation of the object is also key to building a successful intelligent agent that can interact with the environment.

We detected the insect in our project with the help of AprilTags, which simplified the detection and pose estimation problem. Since pests move at high speeds, even with a simplified detection system, we still need to construct an autonomous agent that can effectively chase and remove the pest in real time. A seamless integration of real-time feedback from the camera and the controller is essential to accomplish this goal.

Since autonomous robots need to interact with the

physical environment with manipulators, the study of manipulators by itself is an important topic in autonomous robotics. We took a geometric kinematics approach to describe and plan the motion of a manipulator. Forward kinematics (FK) calculates the end effector pose when the joint angles are given, and inverse kinematics (IK) is the estimation of the joint angles given the desired end effector pose [2]. We planned to use forward and inverse kinematics to implement control programs to allow the robot to consistently maneuver its arm to the desired location.

Most importantly, when an intelligent mobile robot interacts with the physical environment, it needs some form of spatial knowledge. Without an accurate knowledge of its surroundings, a robot cannot reliably or correctly update its own location in the environment; over long periods of time this could result in large accumulated errors and render the robot incapable of performing its task due to the robot assuming it is in the correct physical location that allows it to interact with the bug when in reality it is not. The most common way for an agent to represent spatial knowledge is with a map. However, building a map of an unknown environment requires the agent to know its exact location, so that it can build a map to that local frame of reference. Meanwhile, knowing where the agent is relies on an accurate representation of the environment, or the map. While it is not possible to do both at the same time perfectly, various math models allow computer scientists to build better models to simultaneously build a map and figure out the current position in real time. In this project, we will be using Simultaneous Localization and Mapping (SLAM) to map the surrounding of the MBot and use walls to corner the insect. Having a working localization algorithm also ensures that the robot does not bump into obstacles and walls inside the environment and damage itself or its surroundings as it performs its primary task.

II. Methodology

The goal of our project is to autonomously identify, follow, and catch bugs in an unknown environment while avoiding obstacle collision with an MBot. This goal can be broken down into a few smaller tasks. Firstly, MBot needs a reliable method to find the bugs in real-time and calculate the pose of the bugs with computer vision. Secondly, the MBot needs to plan a safe path to the bug, which can be achieved with SLAM. Thirdly, the MBot needs to follow the bug and reduce the distance to it with vision tracking and a PID controller. Finally, when the bug is within the reach of the Rexarm, it can smash the bug with robot kinematics. The following subsections discuss our approach to solving the individual tasks.

A. Components List

The standard MBot we based our design on has a Raspberry Pi, a Beaglebone, a lidar, a two-wheeled motor, an on-board camera and a Rexarm, but more components are needed to achieve our goal. Two additional cameras (ELP megapixel Super Mini 720p USB Camera Module with 100 degree Lens) were purchased to act as the replacement of PiCam and a secondary camera to increase the Mbot's field of vision. Camera mounts for the new cameras were 3D printed. We needed to acquire a small remote control car to act as the bug for the robot to chase after. We also placed AprilTags on the bug to allow us to keep track of the remote control car's pose. Wooden boards were used to construct a convex environment for the MBot and the bug.

B. Vision Tracking

For our autonomous robot Tom to chase bugs, it needs a real-time vision tracking system. We are proposing to use AprilTags to mark the bugs for tracking. The on board cameras on the MBot can detect the AprilTags and calculate the poses of the bugs. We can then transform the poses in camera frame to the SLAM coordinate frame. Since the bugs will be moving, we will need faster processing speed for the vision tracking. We are planning on optimizing the CPU usage on the Raspberry Pi to dedicate a core for the vision processing task.

Since the camera provides limited local frame of reference, we need to develop a robust FSM for the different states in vision tracking. For example, the first state should be looking for bugs in the current camera frame. If the program detects a bug, then it should enter the follow state and keep following it until the bug is terminated. Since the bug will be moving, it is possible that the bug will escape the

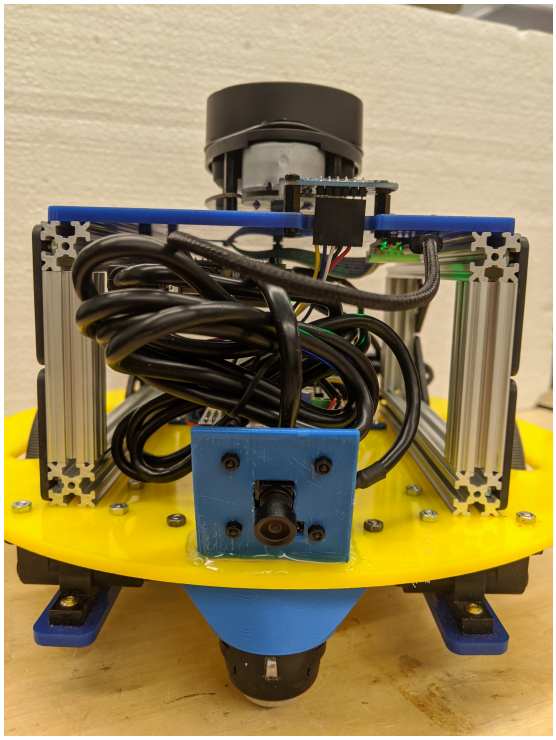

Fig. 1. Camera and mount

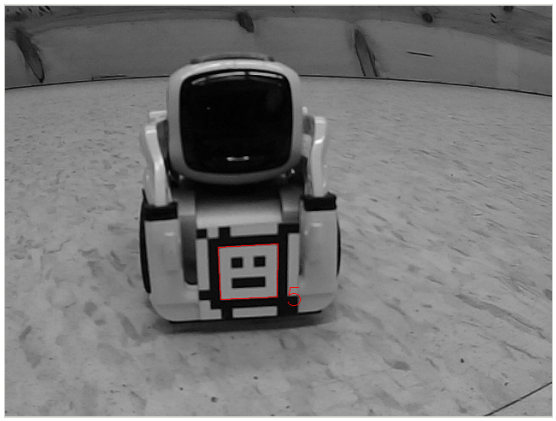

Fig. 2. AprilTag

field of view of the MBot during the vision tracking process. When that occurs, the FSM needs to switch state to turn in the direction of motion of the bug and search for it, or move to broaden the current view to search for a bug in the environment. When the MBot finds a bug, the FSM can then enter the follow state. The FSM is visualized in Figure 3.

One concern regarding vision tracking is whether one camera is enough to provide an adequate field of vision for bug tracking, especially when the bug will be travelling at moderate to high speeds. To address this, we considered mounting two cameras on the Mbot, both facing the robot's heading. However, since we are using both the on board Pi camera and the purchased USB camera, there is a significant difference in video quality and update rate between both cameras, which can make syncing the image provided by both cameras difficult. Moreover, this configuration also did not yield as much of a range increase as we first expected.

After experimenting with other camera configurations, we ultimately decided to mount two purchased USB cameras on the robot: one in the direction of the robot's heading, and the other on the back of the robot, directly opposite the first camera. This will allow the robot to have a doubled field of view that covers a large portion of the one-camera design's blind spot. The two-camera model has the benefit of shortening the searching process and avoiding the bug from circling the robot to avoid detection. The complexity added by the two-camera model is also trivial, as we only need to run the same coordinate transformation as that of the one-camera model and negate the sign of the x coordinate (which corresponds to the heading) and add the robot radius to convert the second camera frame to the world frame.

An issue we ran into when adding two purchased USB cameras is the lack of mounts to secure their locations. Having physically secure cameras that do not move in relation to the robot while the robot is in motion is critical as the coordinate transform is extremely sensitive to camera displacement (since tuning of the camera extrinsic is done at a fixed pose). We decided to 3D print mounts for two cameras. In our first iteration, we used hot glue to affix the camera to the printed piece, however, the heat of the glue damaged the circuitry of the camera and we had to buy a replacement and reprint a frame that uses screws to affix the camera.

C. PID Control

We used a Proportional Integral Derivative (PID) controller for this project. A finely-tuned PID controller can quickly stabilize the Mbot velocity at its desired set point with minimal overshoot and maintain its desired velocity. PID control is an adaptive controller that determines motor commands

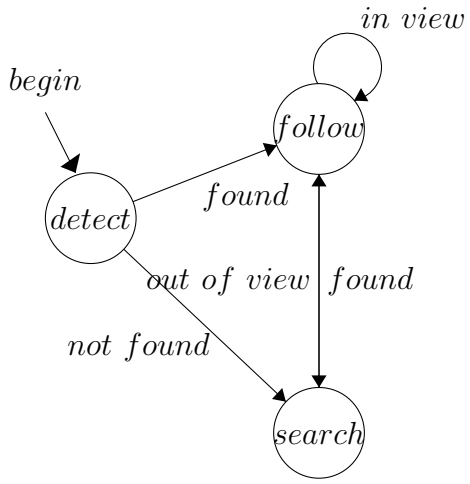

Fig. 3. Finite State Machine of MBot Camera Vision Tracking System

based on feedback. The system has three possible behaviors. When the system is under-damped, the motor velocity oscillates but with decreasing amplitude, asymptotically converging to the desired set-point. When the system is over-damped, the velocity changes slowly to the desired set-point without overshooting. Since an overshooting controller can cause the MBot to bump into its surroundings in a cramped environment and cause damage to the hardware and the environment, we will attempt to create a critically damped or slightly under-damped controller. The reason an over-damped controller will not be used is because the rise time for the output of an over-damped controller is too long. This will make the MBot respond too sluggishly for our needs. The plant we want to control is the velocity set points of the left and right wheels of the MBot. The actuator of this plant is the wheel motor, and the sensors will be the on-board camera, the wheel encoders, and the lidar.

D. Feedback Sensor

We used the cameras on the robot to provide feedback to the robot about the pose of the bug as it moves around the environment. Using this data, our feedback controller will adjust the goal of the robot to give it the best chance of catching up to the bug. The main difficulty should be determining the algorithm that the robot will use to evaluate the best possible position to be in to trap the bug. Another issue that will need to be solved is ensuring that the feedback controller is getting information fast enough from the sensors in order to accurately determine where the bug is so the controller can provide the best possible set of instructions.

E. Motion Planning

When the robot spots a bug and its corresponding AprilTag, it will generate a goal post for the robot to go to. The way the robot travels to said goal post will be determined via the A* path planning algorithm. The algorithm uses the occupancy grid built up by SLAM to calculate the most cost-effective path to the target.

Our high level subsystem integration used a Python control station to perform the decision making, which in turn used C++ code to actuate the plant. The control station would know everything about the MBot's state and what it should do next (search for bugs, drive to bug). If it decided to drive, it would compute the end goal and send it to the C++ subsystems. The C++ subsystems would receive this goal, assume that the MBot wanted to move, and plan out the path via A* planning. The C++ subsystem would also handle the motor actuation and the PID loops that drove the robot along the path.

We chose to integrate our SLAM lab this way because it didn't involve significant changes to the Python and C++ subsystems. The only change we had to make was establishing the LCM connection between the control station and motion planner. In this LCM connection, the control station would send a state and goal (such as drive to goal or spin in place) to the motion planner, and the motion planner will return a status indicating whether the goal is invalid or the MBot has reached the goal. The C++ slam pose also sends the estimated pose to the control station so the station can take the MBot's position into account when making decisions. We went into details about the integration challenges we faced in the Results section.

F. State Machine

The state machine we will implement consists of three major states. Initially, the robot enters the detection state and identifies any AprilTags present in the current camera frame. If no bug is found, the robot will enter the search state. In the search state,

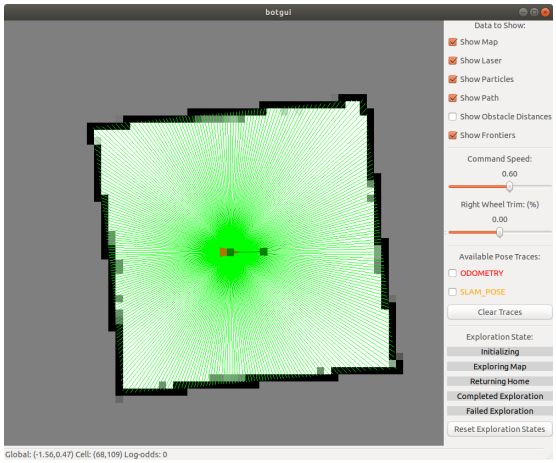

Fig. 4. SLAM

the robot makes use of our convex environment assumption and rotates in place in order to look for the bug. This assumption removes the need for physical exploration during our search process, which conserves our limited on-board battery and simplifies the operation. This is possible because there are no obstacles obstructing the robot's vision or lidar sensors. If no bug is found, the robot will continue its search until one is found.

If a bug is spotted in either the detection or search states, then the robot calculates the position of the bug in the world frame and generates a safe path towards the bug. Once complete, the robot switches into follow mode, where it will travel to the goal pose set by the previous state while constantly updating its goal based on the current location of the bug. This state will persist until either the task is complete or the bug moves out of view, in which case the state machine will return to the search state.

The finalized state machine was able to search for the bug and drive towards it while the bug is stationary. Our next step will be testing and observing what changes we need to make to make the robot chase a moving target.

G. Arm Control

Initially, we planned to use the arm to smash the bug when the Mbot is in range. However, due to issues with the Rexarm, we had to add another intermediate goal before trying to smash the bug. We first tried to run over the bug while it's moving before implementing the smashing functionality. This gave us a good stepping stone before implementing the advanced feature.

The Rexarm presented a lot of trouble. It threw a non-trivial amount of errors due to connection issues, which crashed the GUI. We have removed all Rexarm code and removed the Rexarm from the robot, thus altering our goal of smashing the bug to running over the bug.

H. Robot Target (Mock Bug)

For the Tishman Hall Expo, we built a remote controlled vehicle that served as the bug we are chasing in the demo. This addition to the project provides an added layer of interactivity in the expo and allows us to test the robustness of our design. A video the bug can be found here: https://youtu.be/zOqgw_w2BD8.

III. Schedule Planning

The total time spent on each component of the robot is allocated as follows:

TABLE I
TIME ALLOCATION.

Task	Duration
System Design	Oct. 31 - Nov. 3
Building System	Nov. 3 - Nov. 5
SLAM	Nov. 5 - Nov. 7
Object Detection	Nov. 7 - Nov. 12
Path Planning	Nov. 12 - Nov. 20
Decision Making	Nov. 20 - Dec. 3
System Integration	Dec. 4 - Dec. 8
Poster Creation	Dec. 9 - Dec. 11
Tishman Hall Expo	Dec. 11

IV. Discussion

Our final robot is capable of tracking and following a target traveling in a straight or winding trajectory. It also features adaptive searching, where the robot determines which direction to rotate towards in order to search for the bug based on which direction the bug exited the camera frame. A video demonstration of our robot can be seen here: https://youtu.be/RDEWYEnnFkY. The success of our project will be determined by the following criteria:

A. Ability to Detect Bugs

One metric for determining how well our system worked is its ability to detect the bugs in the environment. In the ideal situation, the program should be able to find all the bugs in the environment. There should be a variety of scenarios that it can react to; here are some examples: the bug is initially in view of the camera, the bug is initially outside of the view of the camera but within the range of the mapped region, the bug is hidden behind an obstacle, and the bug escapes the view of the camera during a chase. A successful robot should be able to respond to most of the above scenarios.

Currently, our implemented search function is capable of addressing all of the presented scenarios except the case where the bug is behind the obstacle. However, since that scenario can only occur in a non-convex environment, and our testing and demo are intended for convex environments only, this is not relevant to our task.

B. Responsiveness to External Stimuli

An important aspect of an intelligent agent is that it needs to respond to changes in the environment and act on it correspondingly. This can include changes to the bug ranging from simple movements to the addition or removal of the bug from the environment.

Currently, the state machine is robust enough to transition and maintain the search state in the absence of bugs, even when we manually removed the bug from its field of vision.

C. Ability to Avoid Obstacles

During the expo, the robot must not be intrusive in its inherent behavior. Therefore, a key part of the evaluation will depend on how capable the robot is at avoiding colliding into passersby and focusing on the task on hand even if the visuals from the camera are cut off by objects obstructing the MBot's view. A robot that damages property and hurts humans will be useless in the real world.

With our assumption of convex environments, the only obstacle present is the arena wall surrounding the robot. Our robot is capable of avoiding these walls during its chase.

V. Possible Follow-Up Projects

There are many improvements that can be made to our prototype to advance it into a functional and commercially-viable product that can address the problem described in the Mary Markley dorm.

A. Functionality in Non-Convex Environments

Our robot is designed under the assumption that it will operate in a convex environment. This assumption provides a convenient simplification to our problem since in convex environments, there are no obstacles, meaning that the robot has a clear, straight-line view of its target (in this case our bug). This simplification not only removes the need for our robot to have a patrol function (instead relying on spinning in place to scout for bugs), it also removes the need to implement obstacle avoidance control when chasing the bug; the latter is a major contributor in the complexity reduction of our A* algorithm, enhancing our speed performance greatly.

However, this assumption does not hold in practical application, as most residential areas will have people and furniture that both obscures vision and obstructs motion. As such, a more sophisticated implementation of our prototype should take these environmental factors into account. Additionally, due to the nature of dormitories, where each tenant has their own, possibly locked, rooms, there would either need to be Mbots deployed for each room (financially infeasible) or the structure of the dorm needs to be altered so the Mbot has free access into each students' rooms (structurally difficult).

B. Removal of AprilTag Assistance

Our robot uses AprilTags to inform the state machine that the object in the camera frame is a bug. This significantly reduces the complexity of vision. With AprilTags, we don't have to worry about computer vision techniques associated with object detection, which is a non-trivial task in vision and removes the need for algorithm construction or the infeasible (due to hardware) machine learning. The AprilTag package also provides a convenient library that translates the tag's position in the camera frame to that of the world frame. This means that by using AprilTags, we avoid dealing with the semantics and localization of our target.

In practice, however, bugs do not inherently carry AprilTags on their bodies. This means that to effectively integrate this robot into its intended environment, we need to employ proper object detection and localization techniques. This could also entail an upgrade in hardware, as our current hardware (the Raspberry Pi 3 and Beaglebone) do not have the computational resources to perform these more complex tasks.

C. Added Dimensionality

Currently, our prototype operates in two dimensions (the ground being the planar environment). This restriction is in part due to the structure of our robot, which is two-wheeled. While it can serve as an effective tool for cleaning up bugs on the ground, it will encounter difficulties reaching for bugs that are at a higher altitude (whether on the walls or in the air). An improvement can be made to address this deficiency. The issue, however, is more complex than it first appears.

In order to properly implement this feature, we will have to perform 3D mapping of the environment in order to properly avoid object collision. This will make both mapping and exploration more computationally expensive, which could mean that the robot will not be able to respond in real-time. Additionally, we will have to modify the physical structure of the robot in order to allow the end effector to reach targets in this higher-dimension environment. The state machine for searching for the bug will also need to be modified and a new camera scheme will need to be implemented to account for the third axis and properly scour the environment.

D. Arm Improvement

One issue we ran into in our design is the extremely faulty arm. Not only does the arm throw exceptions in the code that prevent the state machine from executing, it also generates invalid garbage values for the variables associated to the arm parameter. The presumed cause is that the motors that make up the arm think that the hardware is being damaged because too much torque is being applied; this happens when a lower servo (near the base) or the end effector (the gripper) cannot reach the target angle set by the program (whether it's because the arm orientation makes the base too hard to move or the gripper is holding an object so it can't tighten its grip to achieve the target angle). This is problematic as the arm will never encounter a scenario where the arm motors will be damaged for this reason. And yet turning the assertion (which throws the exception) off will produce undefined behavior resulting in random values in the arm function output. Because of this, we have removed all the Rexarm code and the Rexarm itself and instead modified the robot to be a vacuum-like device that sucks up bugs. However, a fine end effector is necessary when operating in non-convex and 3-dimensional environments in order to reach narrow locations or high altitudes. An improvement in hardware or arm design should address this issue.

VI. Conclusion

By creating a robot that is able to follow and capture bugs, we proposed a solution capable of improving the hygiene and level of cleanliness here at the University of Michigan and finally solve the cockroach issues that have plagued the Mary Markley dorm. This service will be invaluable as student's lives will be immeasurably improved by no longer having to sleep with cockroaches in their rooms. By implementing an autonomous solution, we are able to solve bug problems without great human expenditure and effort along with avoiding potential human error.

VII. Certification

References

[1] N. Papanikolopoulos, P. K. Khosla, and T. Kanada, "Vision and control techniques for robotic visual tracking," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, April 1991, pp. 857–864 vol.1.

[2] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005.

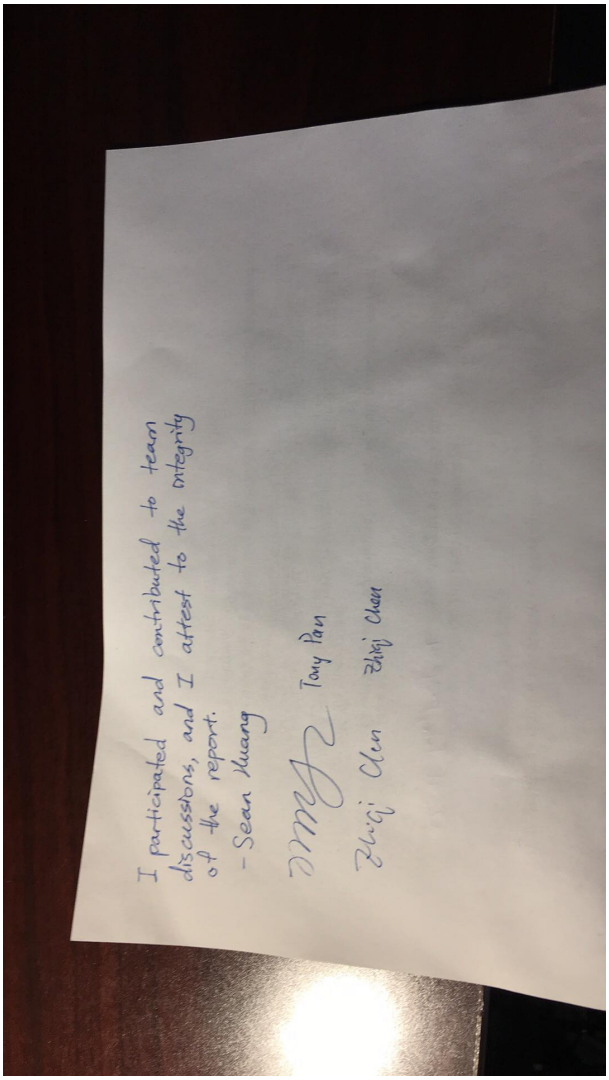

Fig. 5. Certification signed by Sean, Tony, and Zhiqi

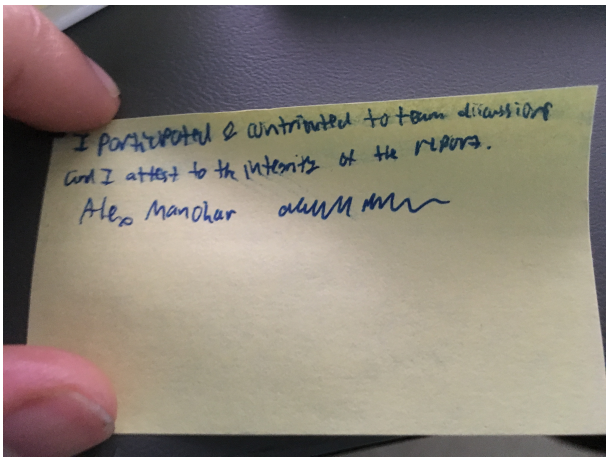

Fig. 6. Certification signed by Alex