EECS 467 Autonomous Robotics Final Report
Grocery Robot Team

Richard Jin, Mingshuo Shao, Zhengyuan Cui, Satat Ojha

I Introduction and Overview

With many elderlies and disabled, it is often hard for them to perform many of the menial tasks that most of the population takes for granted. To combat this, grocery stores have provided alternatives to the normal grocery carts, often making them motorized. However, these devices can be somewhat cumbersome in a grocery store environment with narrow aisles and congestion.

There has also been a surge in the concept of autonomy in the grocery store sector. Giants in China such as Alibaba have implemented stores that are outfitted with Autonomous robots to service growing demand.

We've also decided to try to take a stab at proposing a solution to this problem. We'll be implementing an autonomous driving robot capable of carrying up to 20kg of cargo. The two features we will implement are the setpoint navigation mode and the autonomous follower mode. The setpoint mode will be able to traverse through an aisle to a given location while avoiding expected and unexpected static obstructions. The autonomous follow mode will also be able to track a user's position follows with a safe margin. To accomplish these two tasks, we need to solve problems in the electronic layout, odometry, motion controller, simultaneously localization and mapping(SLAM) as well as person detection and tracking.

Figure 1: Left: current motorized grocery car. Right: Concept of a two wheel version shopping robot

II Description and Scope

A. Scope of the project

1. Assumptions

Due to the complexity of the project, assumptions must be made in order to complete it in a timely manner.

For the navigation mode:

- No dynamic obstructions will be in the map.
- Static obstacles introduced will still allow for traversal.

For the follow mode:

- Only one dynamic-human that is going to move.
- Human will move at a reasonable speed to maintain view in frame.

2. Constraints

a) Budget:

Due to the fact that this is an course project within an university, so there is some limitation on the budget. We ordered these parts online:

(a) 4 x $15.8 H-bridges that compatible to drive wheels on the Magicbot with an operating voltage of 24V (Plan to buy Cytron 10A Bi-directional DC Motor Driver).

Total cost is $63.2 without tax.

b) Resources we used:

(1) One lab laptop for processing the MicroSoft Kinect data and one BeagleBone.

(2) We used 3rd party libraries for ROS and lcm to achieve advanced functionalities, such as openNI, ros2lcm, etc.

(3) We used open-source existing code for YOLO v3 and Kalman Filter Tracking.

c) Time:

This project spans a time period of 8 weeks after the proposal was due (59 days, in which 21 days are weekends or school break). We finished four milestone updates and made a promotional video before the Project Expo on April 24 (details of each milestone explained in section III)

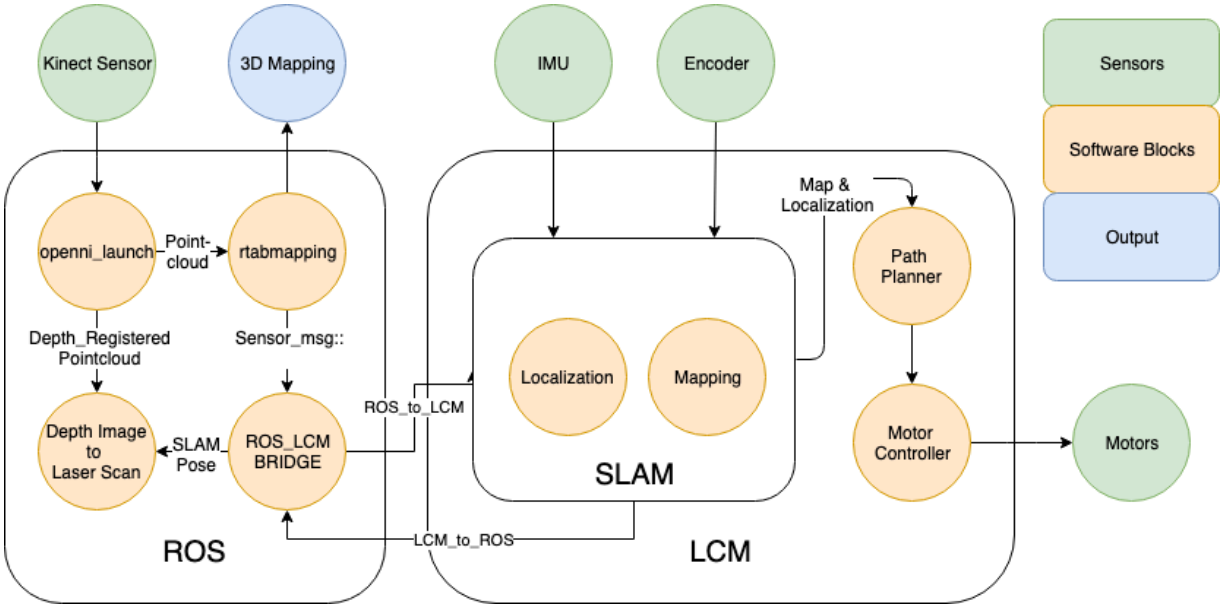

Figure 2: Software Flow Diagram

B. Implementation

1. Software

a) ROS vs LCM

In robotics applications, ROS and LCM both serve as frameworks to communicate between nodes. At first, we came in with the approach of wanting to select one over the other but realized that the better solution would be to use both. Our approach to the situation is to reuse as much of the framework of the Mbot system for driving, odometry and basic framework to free up our time to code a visual SLAM implementation and person tracking features. We ended up deciding to go with ROS due to the existing repositories that have already been implemented with the Kinect. However, to accomplish this we'll need to get a bridge to convert ROS messages to LCM messages and vice versa.

b) Localization and Mapping with rplidar

(1) Baseline: Similar to the Escape Challenge

(a) In a static environment, the Raspberry Pi should work with Beaglebone running odometry, motion_controller, slam, and mobile_bot to achieve SLAM which creates a map of the store.

(b) With only one people walking after it, the robot could navigate to a location send by the laptop.

(2) Reach: The robot is able to localize itself and navigate with at most 1 people interferes with its planned path.

Once the rplidar observe the obstacle in its path, it first stops and scans the environment with rplidar while updating the map. If the map is almost constant (with few uncertain cells) in front of the robot, then it generates a new path and continues navigation. If the robot detects the object getting close to itself, it needs to back out.

c) Localization and Mapping with Kinect

As one of the most challenging part of this project, we didn't use rplidar as the usual SLAM tool. We purely rely on the Kinect for visually sensing the world. More specifically, we uses Kinect's depth sensors to fake 1D laser beams for running SLAM on a 2D plane. We didn't use 2D images for SLAM because we realized that 3D SLAM was too intensive for what we actually needed quite early.

Similar to the Escape Challenge. In a static environment, the laptop should work with Beaglebone running odometry, motion_controller, slam, and mobile_bot to achieve SLAM which creates a map of the store. With only one people walking after it, the robot can navigate to a location send by the laptop.

The algorithm implemented in this section would be the same as the Escape Challenge. However, we need to change the parameters in the odometry and motion_controller to consist with the MagicBot.

Our Kinect is able to produce a 3D reconstruction map of an environment by running RTAB-mapping method on ROS, in which we give the pose of the robot from the result of current SLAM with Kinect data to create a 3D map..

RTAB-Map is a RGB-D SLAM approach with real-time constraints. It uses the bag-of-words approach to continuously determining if the new image comes from a new location or forms a closure. (2*)(3*)

d) How to identify and track the user with Kinect.

(1) Train a Convolutional Neural Network (or use pre-trained as our case) for detecting objects including person so that we can recognize the person from any angle, back and front, partially occluded or not. We used YOLO v3 for simplicity(4*). Possible other choices of existing network include AlexNet, VGG19, etc.

(2) After careful comparison of tracking algorithms, we decided to use Kalman filter because its high tolerance to occlusion. Tracking is modified from Musk Zidan's Kalman Filter. (5*) The result of tracker is explain latter.

e) Path Planning

(1) Given the most recently updated map, we modified implementing the A-Star algorithm to get the robot to the desired destination

(2) We will also use path planning for following the person around, where the destination (i.e the person) will be updated as the person moves. This function has not be implemented due to the short of time.

2. Hardware

Most of our plans for refitting the MagicBot are accomplished. Along with the magic bot, we will need some additional pieces of hardware. The spec are list here:

Hardware	Justification/Use
Cytron MD10C Motor Controller	This motor controller is capable of outputting 5 to 30 V which satisfies the 24-volt brushed motors. They can also drive at 8 amps continuously.
Kinect V1 Sensor	The Kinect V2 sensor has a 70-degree horizontal and 60-degree vertical frame of view. There is a 30 fps refresh rate which is sufficient for both our applications (1*)
Laptop	A laptop will be responsible for running the computationally heavy computer vision and SLAM calculations. The laptop we used enquiped with Intel 7700HQ 16G

	CPU and Nvidia GTX 1060 6G GPU.

Besides of additional hardwares, we also figured out many inner parameters on the MagicBot(Generation 1). The Gear Ratio is 96.0, the encoder resolution is 10. The wheel diameter is 0.197m the wheelbase is 0.55m. We also developed and improved the electronic layout among the BeagleBone, laptop, Kinect, motor drivers, Encoders, motors. Meanwhile, we practiced our skills in how to check continuity of two pins and use it for debugging, flashing BeagleBone, etc. A hardware layout is shown in Figure 5.

3. High-level sensor integration architecture

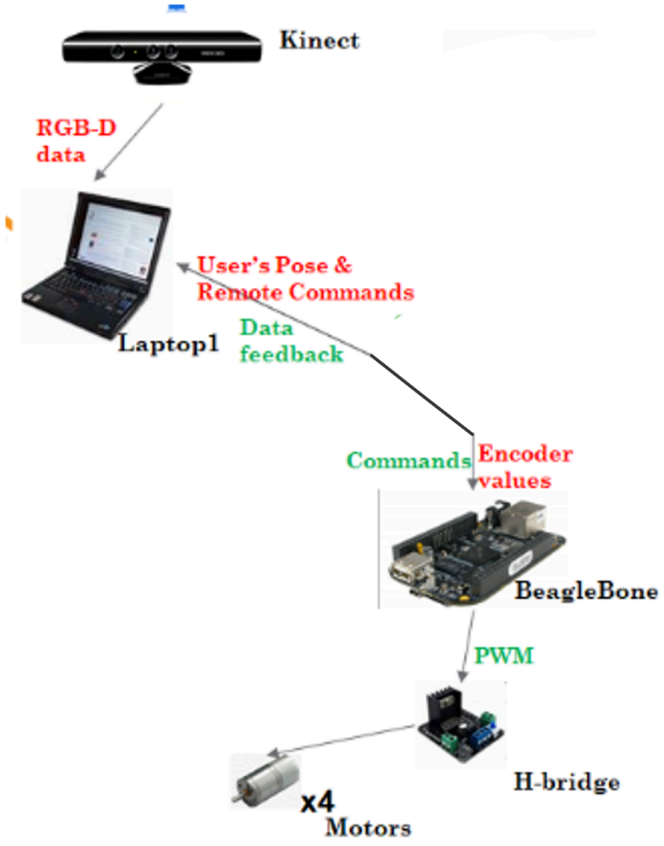

FIgure3: Diagram of architecture

III. Planned Milestone vs Actual Milestone

A. Actual Milestone:

You should use your weekly updates to show how the project evolved from the design stages to the eventual prototype you have delivered.

1. Milestone 1 (3/27):
    - We were able to get Beaglebone Cape for interfacing with various hardware components on our Magicbot (e.g motor controllers, encoders)
    - We were able to understand the hardware components (e.g motors, encoders, fusebox, LiPO battery) in the MagicBot so that we could make the wheels move with its required voltage
    - We set up the ROS environment since we knew we would be using ROS packages in this project
2. Milestone 2 (4/3):
    - Mounted and wired up the different hardware components (battery, fuse, buck converters, motor controllers, etc.)
    - Able to move the robot with control
    - Able to get Kinect data using the ROS OpenNI package, along with “fake” laser scan data
    - Started classier for computer vision
3. Milestone 3 (4/10):
    - Mounted and wired all the hardware components on the MagicBot
    - Figured out the pinouts for the Beaglebone so that we can connect it to our motor controllers and encoders
    - Changed the Botlab code so that it runs on a laptop rather than a Raspberry Pi. This was done by assigning a static IP address and removing all code related to the RPLidar
    - Moved the wheels using the Beaglebone using the drive square code
    - Setup the environment but find the algorithm unstable
4. Milestone 4 (4/17):
    - We were able to make a Beaglebone flasher
    - Figured out the parameters for this new robot (e.g Gear Ratio, Pulses per Revolution)
    - Testing on the MagicBot
    - Setup the environment for the new algorithm we adapted and successfully get the bounding box and relative pose.

B. Problems encountered during the Project

1. *ROS to LCM:*

   Prior to beginning the implementation of the project we had found a repository online that supported the conversion of messages between ROS to LCM and vice versa. However, upon implementation, it became apparent that the conversion only applied to primitive data types. The data type we needed to convert was to convert sensor_msgs/LaserScan.msg to lidar_t.lcm.

   Thus we took it upon ourselves to write nodes in ROS to convert between the sensor_msgs/LaserScan to lidar_t.lcm and pose_xyt_t.lcm to Odometry.msg from LCM to ROS. The Odometry.msg conversion was more difficult as it the two message types were not as similar as an additional step to calculate the quaternion. Example can be seen in figure 4 below.

2. *Odometry theta:*

   Due to using a software differential drive setup on a four-wheeled drive, our turning radius was quite prone to errors due to wheel slippage. To alleviate some of this error, we decided to factor an IMU sensors theta to account for larger turns.

   However, the IMU jittered while the robot remained stationary. Thus a threshold was set to filter out the noise below a certain tolerance. This achieved a certain amount of success for the odometry. However, SLAM was still required to obtain the accurate pose.

3. *LCM packets overflow issue:*

   We noticed that the slam position would freeze for a short duration the mapping phase and resume after a couple of seconds. However, this delay would cause mapping errors to propagate as the laser scans would continue to come.

   We discovered that the main cause of this came from increasing the Occupancy grid size that was being sent over the SLAM_MAP channel and was resolved by passing in the "-R SLAM_MAP" flag into the bot-lcm-tunnel command. This allowed for the packets of data to be kept without delay. This allowed for the beaglebone to avoid subscribing to the channel.

4. *Computer Vision*

   For computer vision, we initially planned to code up an algorithm that was able to identify the person from the input video and get the position of the person relative to the robot and navigate to the place of the person. This part is independent from

all the other software part and was finally found not be able to merge with the other part due to environment issue (python version conflict and gpu installation error). But, we finally achieved the goal of putting bounding box around the tracking person and output the position of the person based on the depth map from kinect. We tried several methods. First, we used Yolo and a tracker. However, the tracker is not stable enough and it often loses track of the person because the person is not near enough to the prediction of the tracker even though the person is actually very near to the prediction in our sense. So, we adapted another algorithm (Yolov3 + Kalmen tracker) and make it only detect human. After solving all the environment issue, it is able to output a bounding box around the person in the sample video. (Figure 5) However, when we try to merge that algorithm into the Ros algorithm, we found that we are unable to do that because Ros is in python 2.7 but out algorithm is in python 3. So, we try to use another Yolo algorithm with similar requirements to try to make it adapt to the environment. However, after several days of trying, we were not able to make it work properly. So, we gave up in the end.

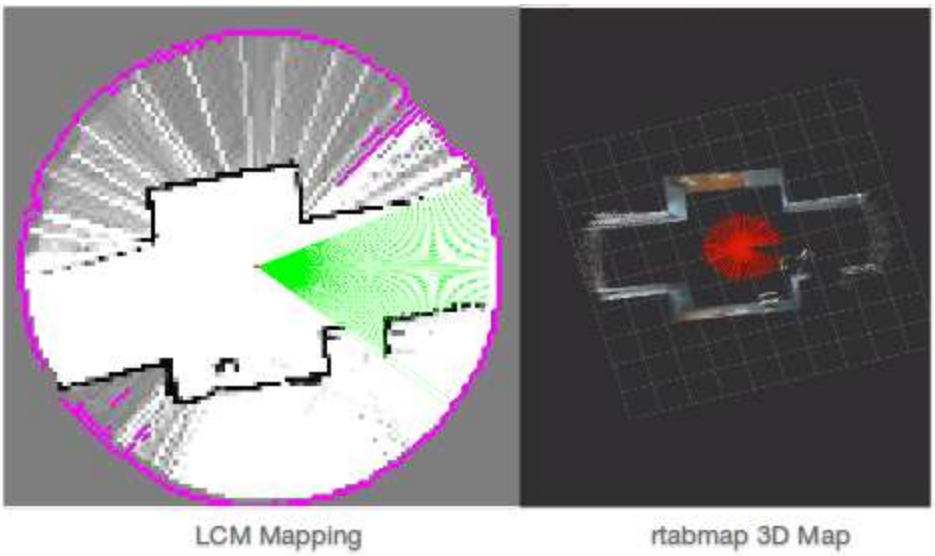

Figure 4: LCM mapping and 3D mapping from Kinect

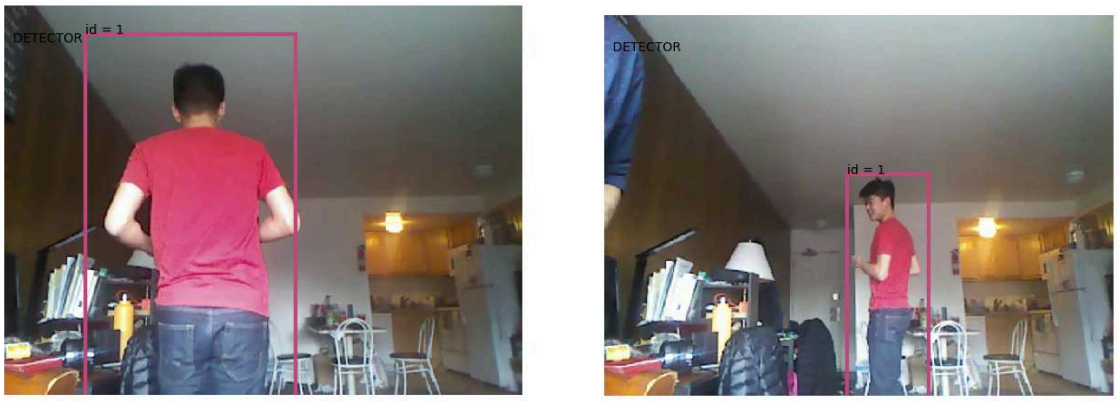

Figure 5; Examples of output with bounding boxes

IV. Results of the Project

We are able to construct a magicbot on our own with given components (figure 5). Our project has following pros and cons:

1. Pros of the current system
   Flattening of 3D objects to allow for the detection of nonuniform objects. In the case of a glass door with opaque areas in certain areas of the door, the Kinect camera is able to accurately detect the distances of those areas. The range of the camera is also further than the rplidar which allowed for further detection.

2. Cons of the current system
   Due to the field of view of the system, the current SLAM lacks the capability of localizing efficiently with the current perception. Also due to the proportional noise generated in further ranges of the Kinect Camera, the accuracy caused a cluster of grid cells to be marked as occupied rather than the intended obstacle. We did not successfully integrated the computer vision into the whole system, which we can definitely do if we have more time.

We made a video demo of the our project which can be find here: https://youtu.be/sDl23mDofeg. Another demo for object tracking: https://youtu.be/ZcJpJs-MKco

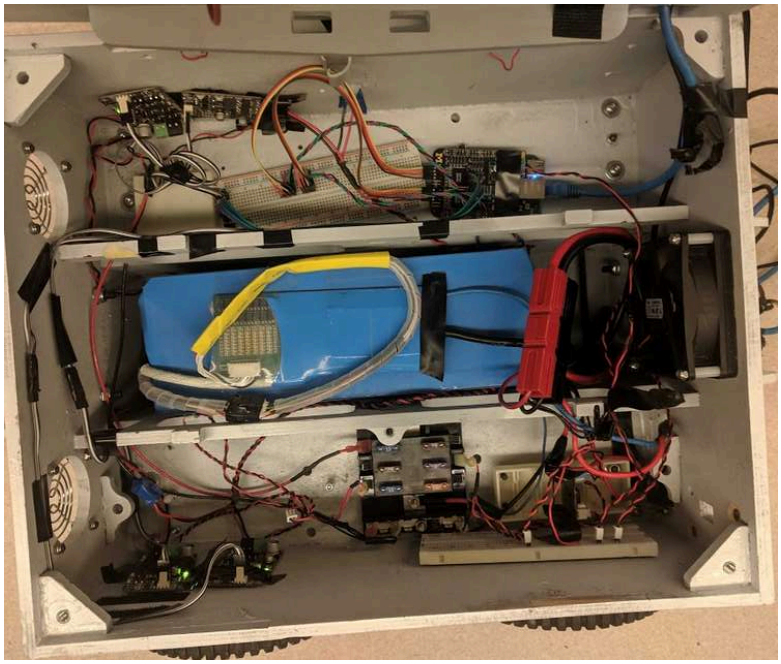

Figure 5: image showing the inner construction

V. Proposal for the BETA Prototype:

I. Movement unreliability
   A. *Problem*: Discovered that due to wheel slippage, the robot had trouble turning in larger than 45-degree increments. Since the grocer robot needed to n 90˚ increments to go across aisles, a different drive system was needed.
   B. *Solution*: A solution is to convert the driving base to holonomic driving. By allowing robots to turn in 90˚ without a change in orientation of the base, the grocery bot is able to more stably move more delicate groceries such as eggs. The main downside to holonomic wheels is having less torque when trying to push objects but this turns out to be an advantage in a densely populated area as it won't be able to push away humans if they accidentally contact.

II. Perception FOV and Range issues :
   A. Problem: Kinect V1 has ~60˚ view with a range perception between [0.5-9] meter. This made it difficult to localize in areas that were not narrow corridors.
   B. Solution: Incorporate multiple cameras with the usage of tf transform trees to increase the viewable range. This would allow the robot to localize with obstacles in mapped in front as it drove through the map. Another possible solution would be to incorporate lidar to allow for sensor fusions to mitigate visual errors in a camera such as an insufficient light source.

III. Integration of dynamic obstacles:
   A. Problem: Did not account for the dynamic moving obstacles such as other customers walking in the store as well as their predicted trajectory
   B. Solution: While radars have difficulty in determining the exact angular distance that the obstacle resides in, the trajectory calculation is accurate due to the unique property of the Doppler effect. This would be useful in avoiding moving obstacles such as other customers.

VI . Reference and citation

(1*)
https://skarredghost.com/2016/12/02/the-difference-between-kinect-v2-and-v1/
(2*)
http://wiki.ros.org/rtabmap_ros
(3*)
https://github.com/felixendres/rgbdslam_v2/blob/kinetic/README.md
(4*)https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch
(5*)https://github.com/ZidanMusk/experimenting-with-sort