Course Number:ME-GY 996X
MS Project Report

Autonomous Ground Robot for Fruit Counting at Orchard using SLAM and Machine Learning

Submitted in partial fulfillment for the degree of
Master of Science (MS) in Mechatronics and Robotics.
by
Jing Xia

Submitted in partial fulfillment for the degree of
To the department of
Mechanical and Aerospace Engineering
(Fall 2017)

NYU | TANDON SCHOOL OF ENGINEERING

Abstract:

The aim of this project is to develop a prototype of a full autonomous ground robot that can achieve three main goals: Simultaneous Localization and Mapping(SLAM), Navigation and Counting the number of fruits using machine learning to estimate the production of orchard to assistant farmers. The robot with these functions could improve farmers' efficiency and save their labor.

The project in this semester is only fit for apple orchard. However, it is easier to change the object type if users have their dataset and training the machine learning model. The SLAM and navigation part should be compatible for most of the flat ground once the map is build. On this project, all the testing is in inside the lab but with real apples.

Index

Background:

The Simultaneous Localization and Mapping (SLAM) is a key driver behind unmanned vehicles and drones, self-driving cars, robotics, and augmented reality applications. The main idea of SLAM technique is to leave the robot at an unknown location and let it move and build a consistent map of its surroundings.

For the objection detection, it is an important branch in the computer vision. Generally, it required to deal with all three: detecting and classifying and tracking objects in images and videos. Deep Learning revolutionized this paradigm by compiling Computer Vision and Machine Learning into a single field. Deep Learning can classify images from raw pixels using convolutions and deep neural networks.

These two fields are two important parts in robotics and self-driving car field. Every self-driving car companies has equipped the Lidar and camera on their testing car to implement their algorithm.

In 2015, Dr.Vijar Kumar from University provides a concept called precision farming and said that precision farming allows a farmer to get input on every tree in an orchard to know if water, fertilizer or pesticide are needed. From his TED talk, it shows that the drone that can finish the precision farming has two have two basic function which are SLAM and objection detection.

However, the battery of the drone will be an issue since most of the drone fighting time is less than 25 minutes. As a result, the ground robot should be an alternative way to do it also.

Introduction:

In this project, the robot in this project has two basic functions: SLAM and Objection detection based on machine Learning. For the SLAM part, the algorithm is based on the Hector SLAM algorithm only using Lidar. For the Object detection part, the object in this project is apple, which is used in farm or orchard. In addition, the robot could count how many apples per plants so that it can estimate the production of each yield by sampling certain times. In general, the more sample you do, the better result you will get. For the processing image part, in this project, there is not in real-time since the microprocessor on the robot is not powerful to handle these kinds of computing requirement since it also need to process the laser scan messages and do the path planning to do the navigation at same time.

This robot can do SLAM of an unknown environment. In this approach, the algorithm to implement SLAM is HECTOR, which require less sensors but more robust in complex environment features. Once the map is build, the robot then can do the path planning and navigation to the destination using AMCL and Dijkstra algorithm. In this project, all the software is run on the ROS on Linux system. The Linux system is installed on the microprocessor on the robot. And the objection detection function will be flexible based on the users' situation.

For the demo, the robot will draw the map of the orchard and do the localization and navigation to certain destination points and count how many apples on each apple trees then estimate the whole production.

A GUI system will be created and show the map image information in real time based on the ROS QT or ROS Android after all the basic function works.

In the future, the objection detection will depend on the user's environment and it is flexible, which means users can use their own data set to training the algorithm to get the object that they want.

Hardware Design:

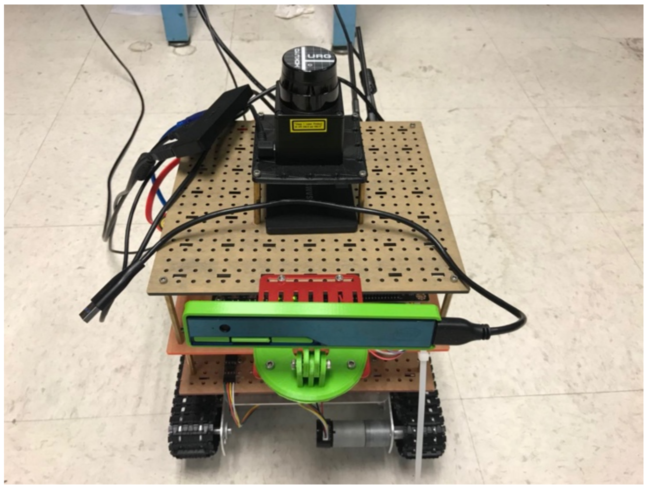


Figure1. Front view of the robot

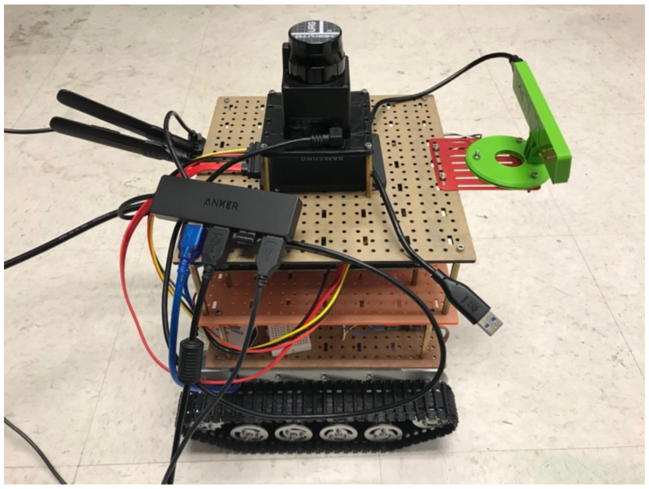


Figure2. Side view of the robot

For the whole Robot, the size of the robot is 270mm* 255 mm*405mm. The weight totally is around 9.2kg without cables.

In the flowing list, it shows all the parts in this system:

Component name	Quantity
NVIDIA Jetson TX2 development board	1
Arduino mega board	1
Vinsic 19V power bank	1
Li-Po 3s battery	1
Hokuyo URG Lidar	1
Intel Realsense camera r200	1
DROK Buck and boost voltage regulator	1
Robot chassis with two 9V DC motors	1
Samsung 256G SSD	1
Workbench layer board	3
Anker USB hub with 4 outputs	1
Adafruit Arduino motor shield	1

Table1. Component list of the robot

Parts:
Power Part:

Figure3. 19V power bank

Vinsic 30000mAh
The Capacity of this battery is 30000mAh.And there is a DC output port, which is 19V/3.5A. This could enough voltage and current to the Jetson TX2. Since the current of Jetson could increase to 4A, so the time could be around 7.5 hours.
T= Q/I= 30000mAh/4000mA = 7.5h
In addition, based on the experiment, the robot ran around half hour and the battery decrease 7%.

Venom 20C 3S battery:
Battery voltage is 11.1V and battery capacity is 2200mAh.
Based on my calculation, it could provide 1 hour battery for both motors
T= Q/I= 2200mAh/2200mA = 1h
The battery is connected to the Arduino Mega and provided power to the motors.

Figure4. 3s battery

Microprocessor:

NVIDIA Jetson TX2:
NVIDIA Jetson TX2 Development board is the master for the whole system, which will process all the sensors information and send to the slave. The TX2 is powerful compared to Raspberry pi other types of controller. It has NVIDIA Pascal™ Architecture GPU and 2 Denver 64-bit CPUs + Quad-Core A57 Complex. It also has 8 GB L128 bit DDR4 memory and 32 GB eMMC 5.1 Flash Storage. In addition, it also has USB 3.0 Type A, USB 2.0 Micro AB (supports recovery and host mode) HDMI, SATA Data and Power, GPIOs, I2C, I2S, SPI. In this project, it also connects to a 256GB SSD to storage data.

Figure5. NVIDIA Jetson TX2

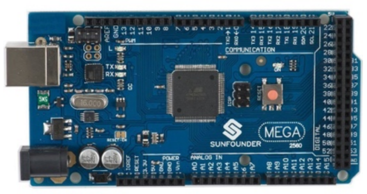


Figure6. Arduino Mega

Arduino Mega:
Arduino Mega board is the slave of the whole program, which will receive the message from the master by USB serial communication. The reason to choose mega is that mega has more Digital I/O pins (54) and Flash memory(128KB) compared to Arduino UNO, which is more compatible to this project.

Arduino Motor Shield:
Instead of using a latch and the Arduino's PWM pins, it provides a fully-dedicated PWM driver chip onboard. This chip handles all the motor and speed controls over I2C4 H-Bridges: TB6612 chipset provides 1.2A per bridge (3A for brief 20ms peaks) with thermal shutdown protection, internal kickback protection diodes. Can run motors on 4.5VDC to 13.5VDC.

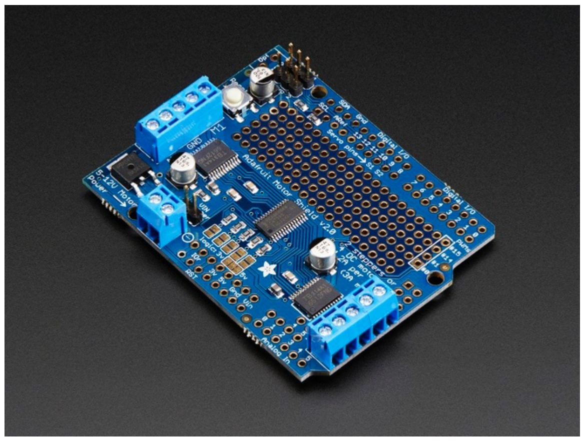


Figure7. Adafruit motor shield

Sensor:

Hokuyo URG-04LX-UG01:
Hokuyo's URG-04LX-UG01 detectable range is 20mm to 5600mm.The frequency for that Lidar is 100msec/scan
The operating voltage is 5V, which is supported directly by USB port from Jetson TX2. The range of scanning is 240° with 0.36° angular resolution. The Lidar will provide the main information on Hector SLAM part and navigation part.

Figure8. Hokuyo Lidar

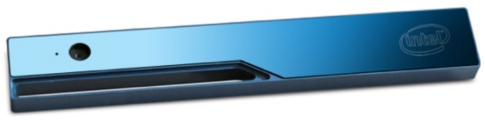


Figure9. Intel RealSense Camera

Intel RealSense Camera R200:
Intel RealSense Camera R200 has depth/infrared and RGB camera. It has its own Intel processor and Graphic unit compatibility to the LINUX system and ROS. The size of it is 101.56mm length x 9.55mm x 3.88mm.

Chassis:

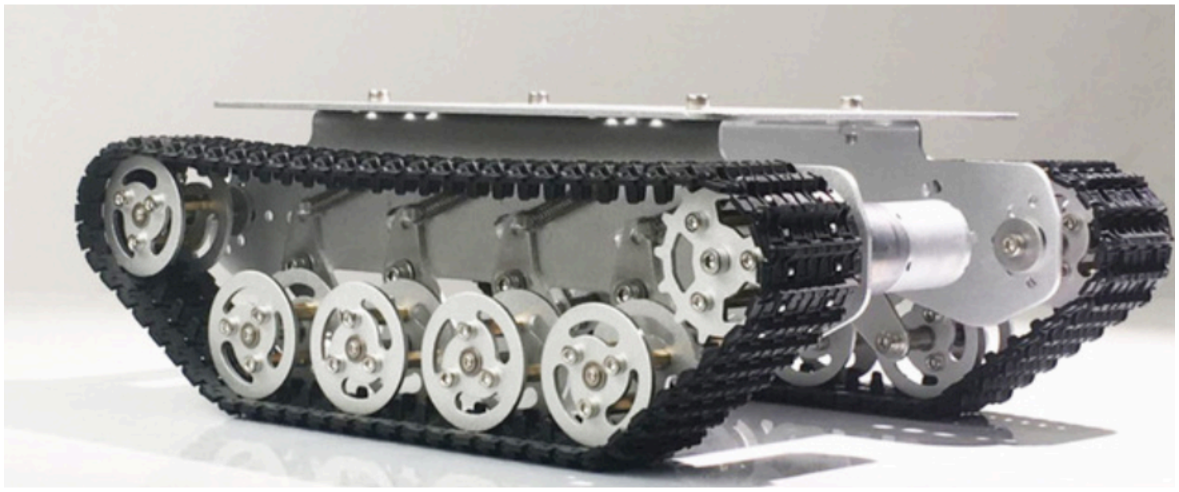


Figure10. Robot Chassis

These chassis have caterpillar instead on the normal wheels. Each driver is connected to a DC motor, the driven wheel each side will be driven due to friction between the track and itself.

Actuator:
DC motor:
Name: 25mm gear motor
Output speed: 150±10%rpm
No_load Current: 200mA (Max)
Stall current: 4500mA(max)
Stall torque: 9.5kgNaN
Rated speed: 100±10%rpm
Rated torque: 3000gNaN
Rated Current: 1200mA (Max)
Noise: 56dB
Working voltage: 9V
encoder: 2 pulses/circle

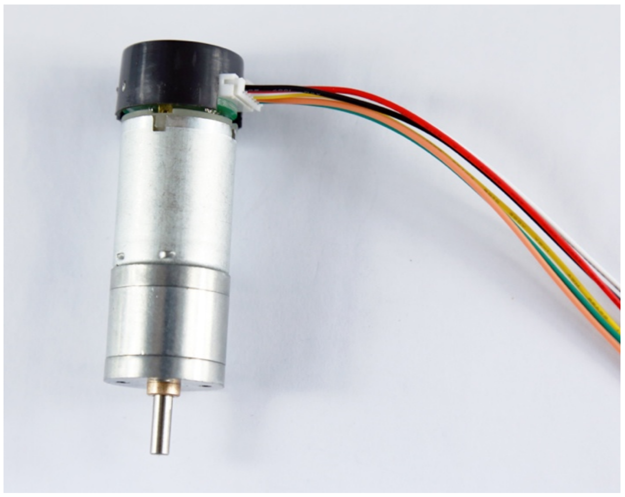


Figure11. DC motor

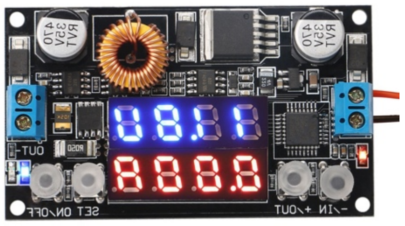


Figure12. Voltage Converter

Others:

Drok Converter
5-32V to 0-30.0V adjustable buck CC CV converter with volt amp display. 0-5A constant current / voltage output, over 2A please add a heatsink. It could buck 11.1 V battery to 9 V, which provides the voltage for the DC motors.

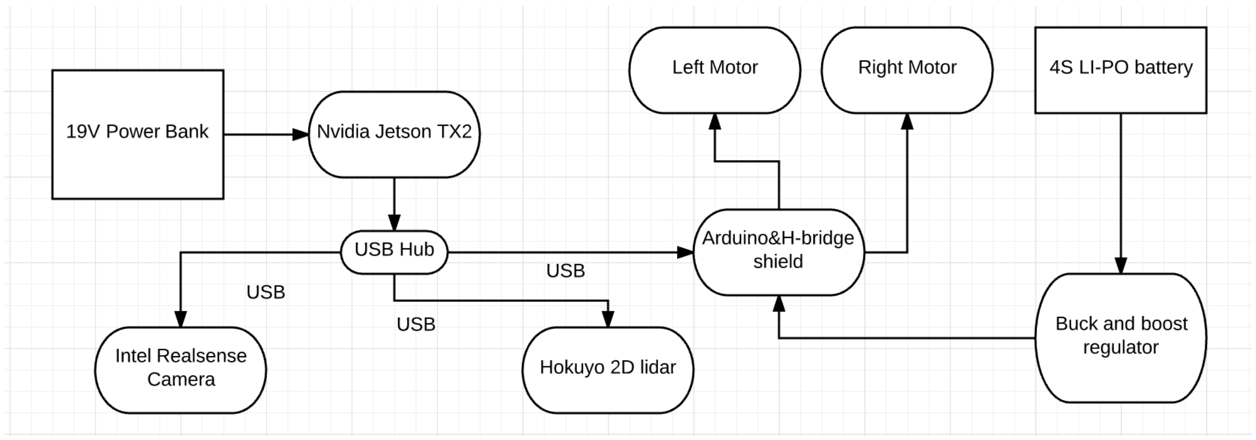


Figure12. Hardware set for the whole system

For the whole hard system, as the flowchart showing, the main processor will be NVIDIA Jetson TX2 which is the master in that system. The NVIDIA Jetson TX2 process the Hokuyo Lidar scan information and Intel RealSense Camera. Then it connects to Arduino Mega by USB serial communication. Since the Jetson only has 1 USB port, it is required at least 3 USB ports (Lidar/Camera/Arduino Mega) in this system. As a result, a 4-port USB hub is used to reach the requirement. Arduino's power is supported by Jetson TX2 by USB 5V.

In the whole robot, it has 3 layers, each layers contains differeent compoments for the whole system. Each layers size is 21 X 22 cm. Each layer, it is connected by skewskid and nuts.In the first layer, Arduino Mega and voltage regulator are put over there. Additionally, the 3S LI-PI battery is also in that layer. In second layer, there are the Jetson and its ssd. On the top layer, it contains the 3d-mounting printed part for camera, and the small suqare part for Hokuyo lidar.

Software Design:

The whole software is on the Linux system. The operating system edition on the Jetson TX2 is Ubuntu 16.04. For Robotic Operating System(ROS) on it is ROS Kinetic. The reason for use ROS is that it is a flexible framework for writing robot software. And it also contains a lot of tools, libraries, and conventions that aim to simplify the tasks.

From the sensor part, it is required to install the hokuyo_node package and Intel_realsense package on the ROS Kinetic.
For the whole system, it can be divided into two main functions:

- SLAM and Navigation
- Take photos and Count apples

SLAM and Navigation part:

In SLAM part, it could be divide into two parts:
There are two main algorithms on it:

- Hector SLAM

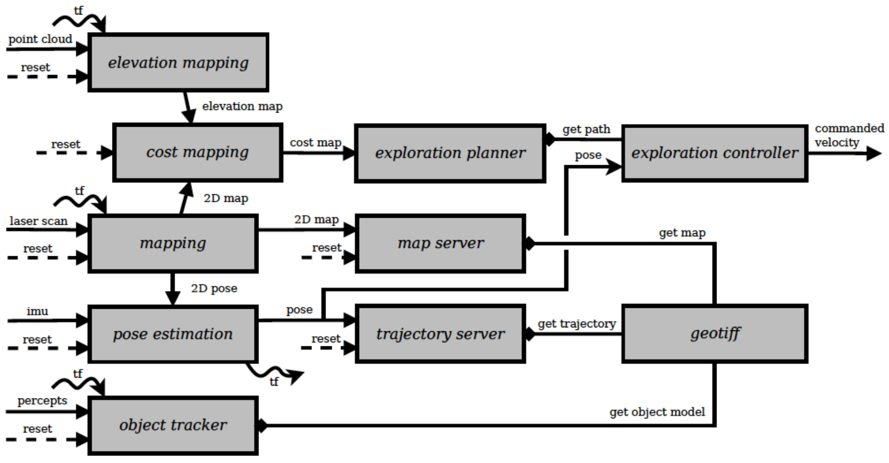


Figure13. Flowchart of Hector SLAM

Hector SLAM is a SLAM approach that can be used without odometry as well as on platforms that exhibit roll/pitch motion (of the sensor, the platform or both). It leverages the high update rate of modern LIDAR systems like the Hokuyo UTM-30LX and provides 2D pose estimates at scan rate of the sensors (40Hz for the UTM-30LX). While the system does not provide, explicit loop closing ability, it is sufficiently accurate for many real-world scenarios. The system has successfully been used on Unmanned Ground Robots, Unmanned Surface Vehicles, Handheld Mapping Devices and logged data from quadrotor UAVs.[1]

- AMCL

AMCL is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map. A key problem with particle filter is maintaining the random distribution of particles throughout the state space, which goes out of hand if the problem is high dimensional. Due to these reasons, it is much better to use an adaptive particle filter which converges much faster and is computationally much more efficient than a basic particle filter.[2]

In this project, it can be divided into several steps:

- Use hector SLAM to build the map of 5th lab first by manual control robot go around the lab.
- Save the map to the map_server
- Use AMCL and move_Base function to let the robot do the navigation part.

The key point for this part is that the robot doesn't have any other odometry to provide the information for the navigation which only use lidar_scan_match function to get the scan information and converted it into odometry information and send to the AMCL.

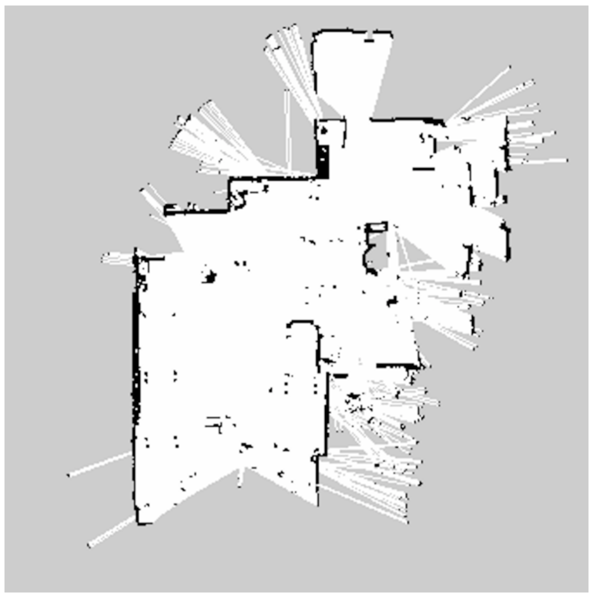


Figure12. map of the 5th lab draw by Hector slam

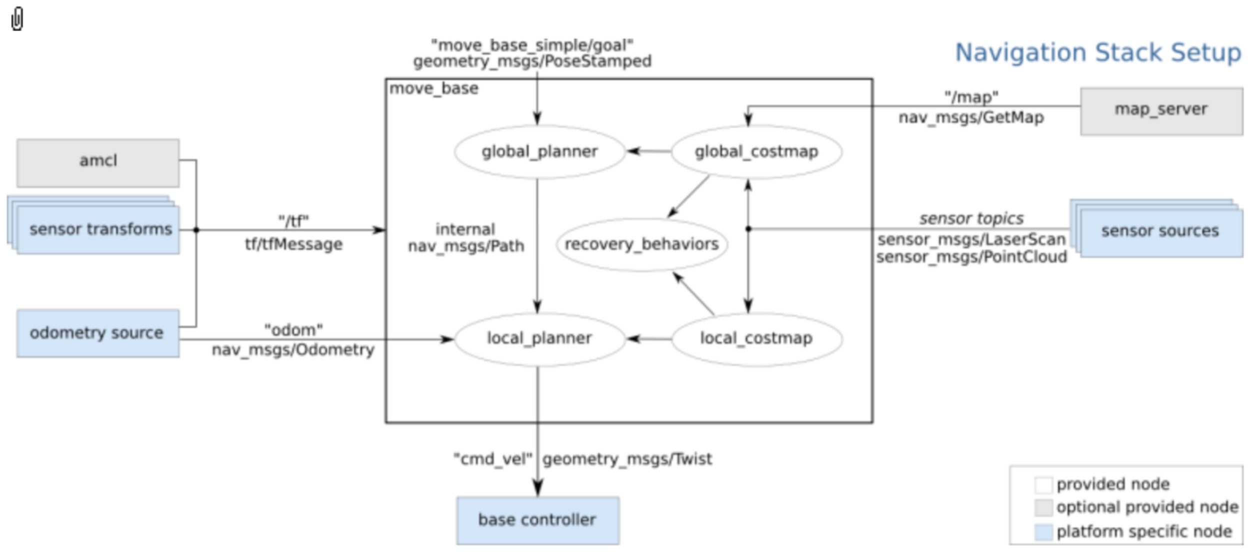


Figure13. General Navigation Stack Setup

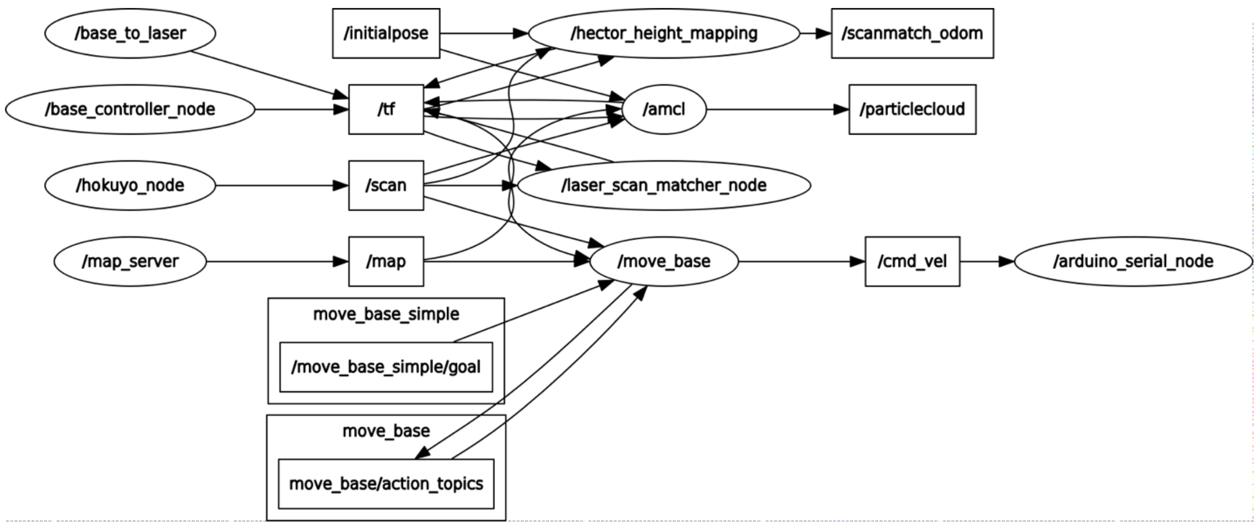
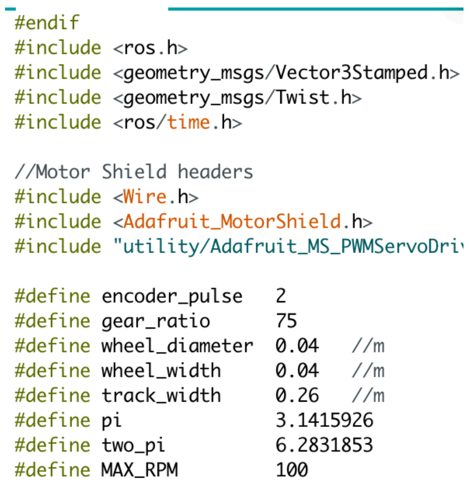


Figure14. rqt_graph of this robot

In this graph, it shows the relationship to construct a complete system to let the robot do the navigation part. In this project, the sensor sources is from Hokuyo Llidar, the odometry source is from laser_scan_matcher_node. For the sensor transforms, it is declared directly in ROS launch file using tf function since the position of baselink and Lidar is always fixed. For the base_controller part, it is written in Arduino code. Generally, it transfers the cmd_vel topic into corresponding information to wheels left and right pwm value. For coding this part, it is required to set up kinematics of this two wheels' robot. The data to for calculating this model, it is required that to know some basic parameters of the robot as following code showing:



```
#endif
#include <ros.h>
#include <geometry_msgs/Vector3Stamped.h>
#include <geometry_msgs/Twist.h>
#include <ros/time.h>

//Motor Shield headers
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWMServoDri

#define encoder_pulse   2
#define gear_ratio      75
#define wheel_diameter  0.04   //m
#define wheel_width     0.04   //m
#define track_width     0.26   //m
#define pi              3.1415926
#define two_pi          6.2831853
#define MAX_RPM         100
```

Figure15. Arduino code of base_controller

Object detection and Counting:

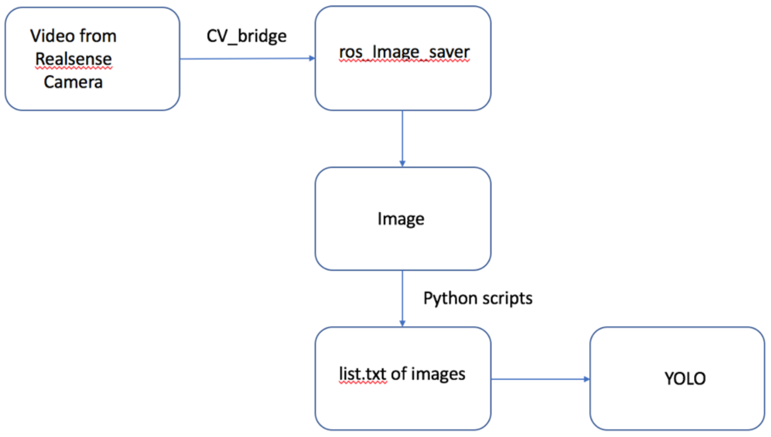


Figure16. Flowchart of functions of taking photos

In the object detection and counting part, the input is image. Since the input from camera on the robot is real-time video, it needs to be converted into image_saver by CV_bridge package. This part is running on the ROS platform. The ros_image_saver package could save the pictures to the current folder by time interval. In this case, robot will take a photo every one second and save it to the folder. In the corresponding folder, a python script would run, which could read and write all the images names into a list.txt file. Then YOLO will run the program to this list directly and detect how many objects in each image and return to its value.

For the YOLO part, you only look once (YOLO) is a state-of-the-art, real-time object detection system. Based on author's paper, the object detection could be thought as a regression problem to spatially separated bounding boxes and associated class probabilities. In YOLO, a single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

For this project, detector.c and image.c files are rewrite for this certain task, then adding a function called draw_detections_apple. In this function, adding a counter to count and accumulate the apple numbers.

Ideal Testing Case:

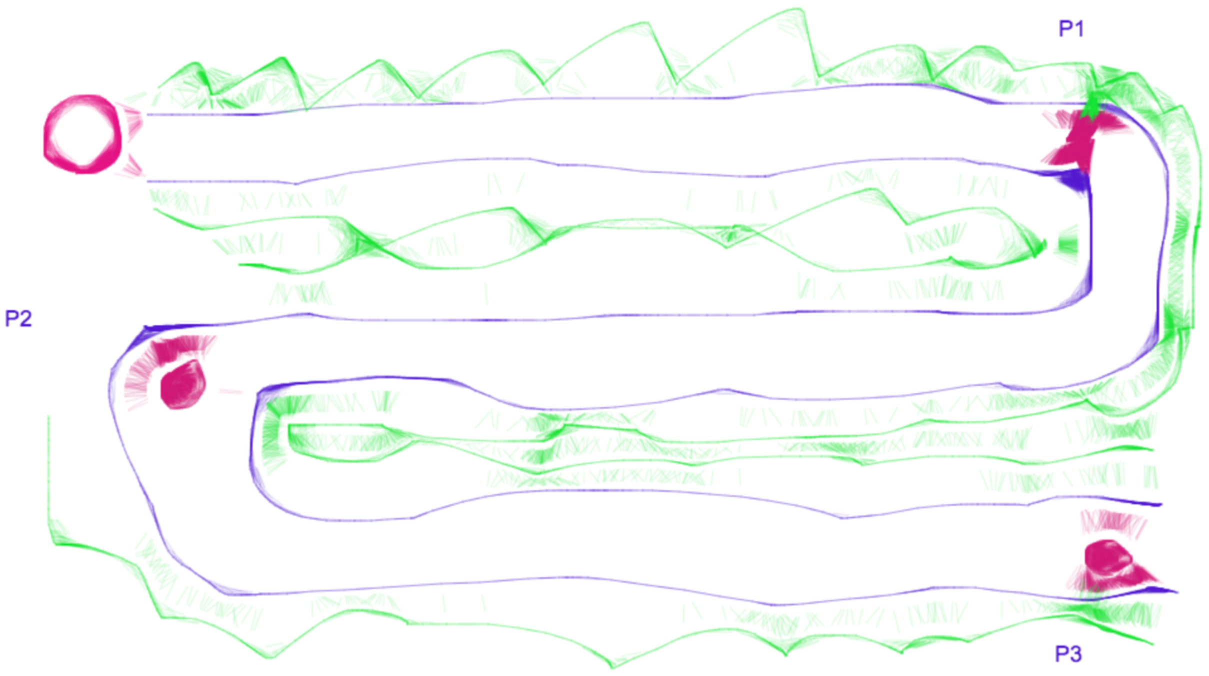


Figure17. Robot Demo sketch map

As the sketch map showing, the robot will set up 4 designations to count the number of the apples of that location, then count the whole number. In the launch file, it declares that the robot will arrive to destination first and take photo. The constrain to control the frequently of taking photos now is just use time interval. For next destination, the robot will stop the camera function when its moving.

Evaluation:

For the evaluation part, the idea is to test the unit function first and test the whole Task function. For the unit function, it could be divided into two parts: Navigation and furit counting.

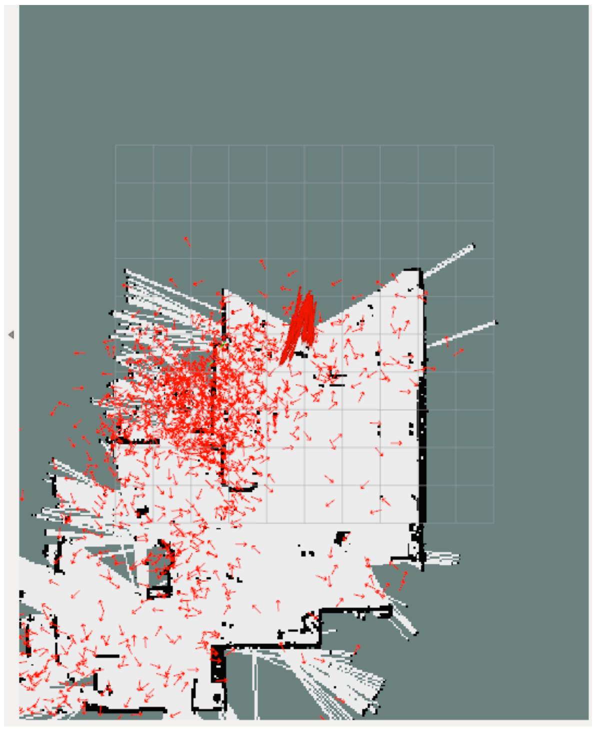


Figure18. Robot Demo sketch map

From the Figure 18, the odometry kept shifting, which showed the particle cloud can't converge to some certain area. It leads to the robot got stuck or kept rotate when it generates the path to the destination point since it kept generating new path when the odometry changed.

The reason for causing this issue could be many possibilities:

- Transform relationship between base_link and Map is wrong
- Odom itself has huge error for scan information due to the launch file parameters
- The robot rotates too faster

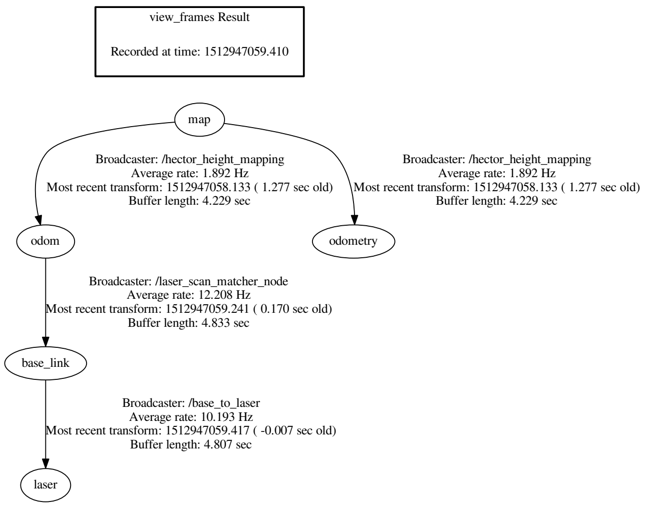

view_frames Result
Recorded at time: 1512947059.410
map
Broadcaster: /hector_height_mapping
Average rate: 1.892 Hz
Most recent transform: 1512947058.133 (1.277 sec old)
Buffer length: 4.229 sec
Broadcaster: /hector_height_mapping
Average rate: 1.892 Hz
Most recent transform: 1512947058.133 (1.277 sec old)
Buffer length: 4.229 sec
odom
odometry
Broadcaster: /laser_scan_matcher_node
Average rate: 12.208 Hz
Most recent transform: 1512947059.241 (0.170 sec old)
Buffer length: 4.833 sec
base_link
Broadcaster: /base_to_laser
Average rate: 10.193 Hz
Most recent transform: 1512947059.417 (-0.007 sec old)
Buffer length: 4.807 sec
laser

Figure19. tf_frames of the whole system

By looking the tf_frames, the odom frame relationship with map is correct. And even the robot stops the odom is still drift. These two results showed the problem may be caused by ROS launch file parameter setting.

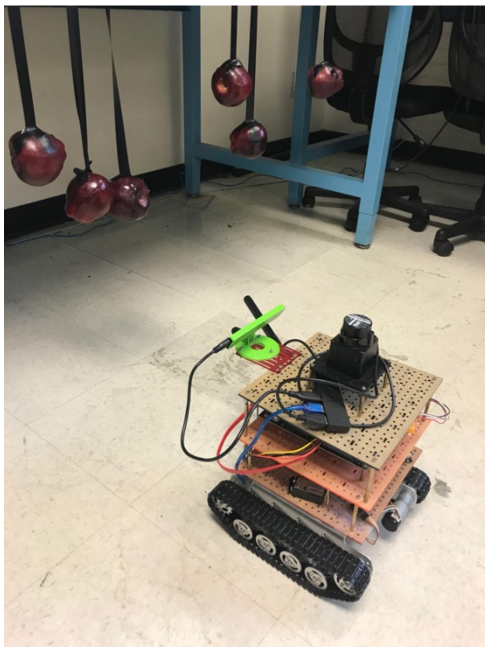


Figure20. Testing environment for count fruiting

Due to time and space limitation of access the real orchard, the demo environment is setting artificially. 6 apples were taped under the desk. And the robot running the function to take photos and save it to the local. After this, users ran the YOLO to counting the fruit. Each output for the input picture is called prediction.

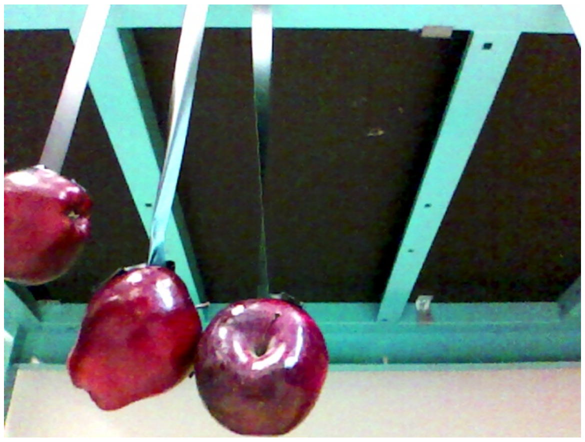


Figure21.Picture1 took from robot

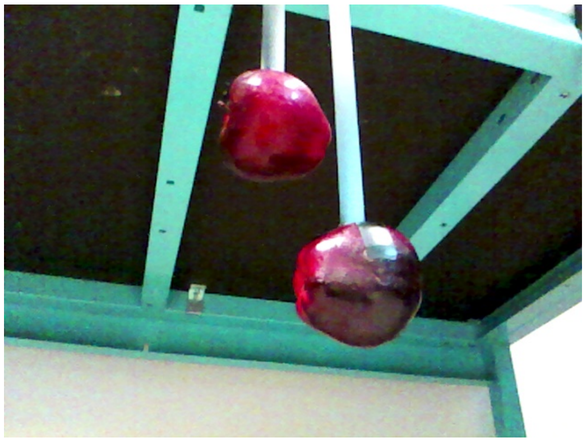


Figure22. Picture2 took from robot

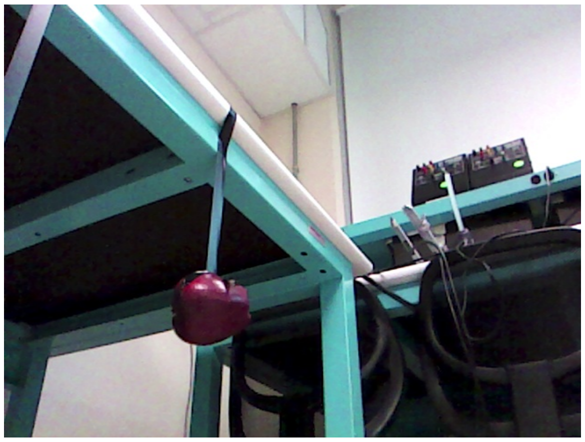


Figure23.Picture3 took from robot

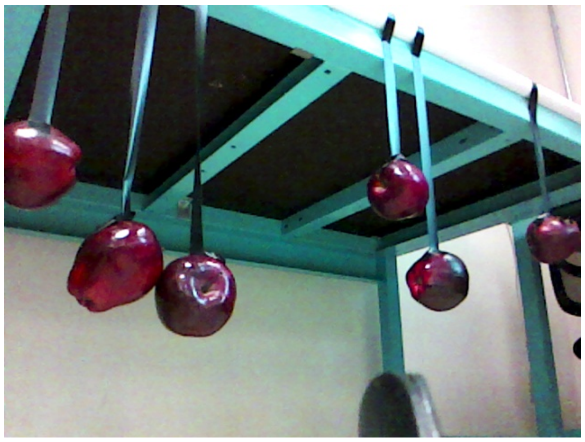


Figure24.Picture4 took from robot

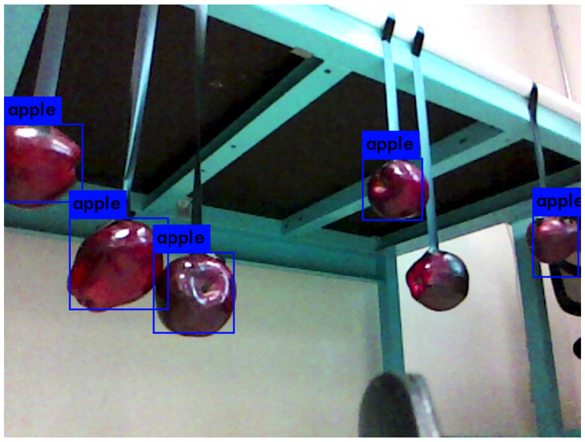
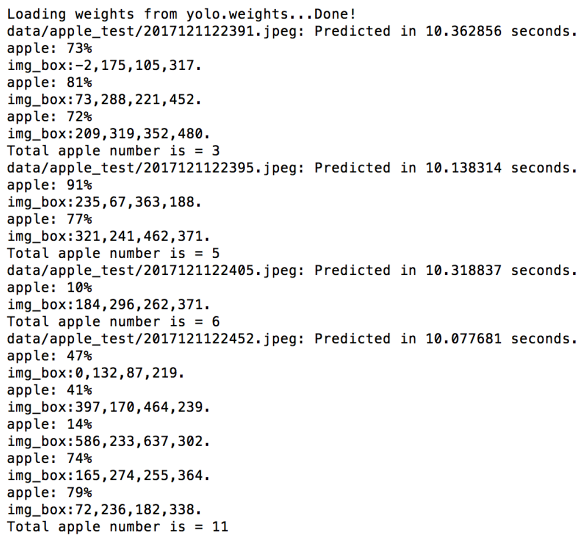


Figure25.Prediction Sample Output



```
Loading weights from yolo.weights...Done!
data/apple_test/2017121122391.jpeg: Predicted in 10.362856 seconds.
apple: 73%
img_box:-2,175,105,317.
apple: 81%
img_box:73,288,221,452.
apple: 72%
img_box:209,319,352,480.
Total apple number is = 3
data/apple_test/2017121122395.jpeg: Predicted in 10.138314 seconds.
apple: 91%
img_box:235,67,363,188.
apple: 77%
img_box:321,241,462,371.
Total apple number is = 5
data/apple_test/2017121122405.jpeg: Predicted in 10.318837 seconds.
apple: 10%
img_box:184,296,262,371.
Total apple number is = 6
data/apple_test/2017121122452.jpeg: Predicted in 10.077681 seconds.
apple: 47%
img_box:0,132,87,219.
apple: 41%
img_box:397,170,464,239.
apple: 14%
img_box:586,233,637,302.
apple: 74%
img_box:165,274,255,364.
apple: 79%
img_box:72,236,182,338.
Total apple number is = 11
```

Figure26.Prediction Sample Output

Based on the demo result, the default threshold 0.2 is not good, which only detects seven apples totally. Then switch to threshold to 0.1, the result becomes 11.

Threshold	Pic 1	Pic 2	Pic 3	Pic 4	All	Correct%
0.2	2	2	0	3	7	63.6%
0.1	3	2	1	5	11	91.7%
Reality	3	2	1	6	12	

Table2. Apple count result

Now the number of default classifier is 20. Based on the testing case, on a desktop with NVidia 1080 Titian GPU, the predicted time could be lower around 0.9 second. On my robot without NVIDIA Compute Unified Device Architecture and calculated by CPU, the Processing time for each picture is around 10 seconds.

Future Plan:

1. Fix the issue of odometry drift when navigation
   In this case, it may require add an IMU or visual odometry to get a pose from sensor fusion. There is a package called robot_pose_ekf.
2. Combine the navigation and fruit_counting function at one multi task function using action_lib ROS package.
3. GUI for the application(Android)
4. Re-training the model use another dataset (more types/faster)
5. More types of data to collect such as humidity/temperate
   Adding humidity and temperate sensor to the Jetson TX2 by I2C connection, and publish this sensor information to certain topic and subscribed it on the GUI.
6. If the terrain is more complex, pitch change
   If the pitch degree changed, the navigation part will be destroyed. In this case, it is necessary to detect if there is a pitch degree change or any change in Z axis.
7. Multi robots count and slam together
   If the orchard square is too big, one robot may take long time to finish the whole task. For some consideration, using multi robots work same time could save time and increase efficiency.

Conclusion:

In this semester, the navigation part, this approach is not successful since the robot can't do navigation properly.
The counting apple function is finished but the speed and accuracy could be improved by re-training the model use some other dataset. For the future, all the file names of the images that robot took could be saved into the text file directly instead of running the python script. The predicted time could be less if decreasing the types of classifier. In addition, the whole system needs to have more experiments to test its robust and variation.

Reference:

1.http://wiki.ros.org/hector_mapping?distro=kinetic
2.http://wiki.ros.org/amcl
3.Redmon, J. (n.d.). Retrieved December 14, 2017, from https://pjreddie.com/darknet/yolo/