Andrew Gates

Professor Nikolay Atanasov

ECE 276A

15 December 2017

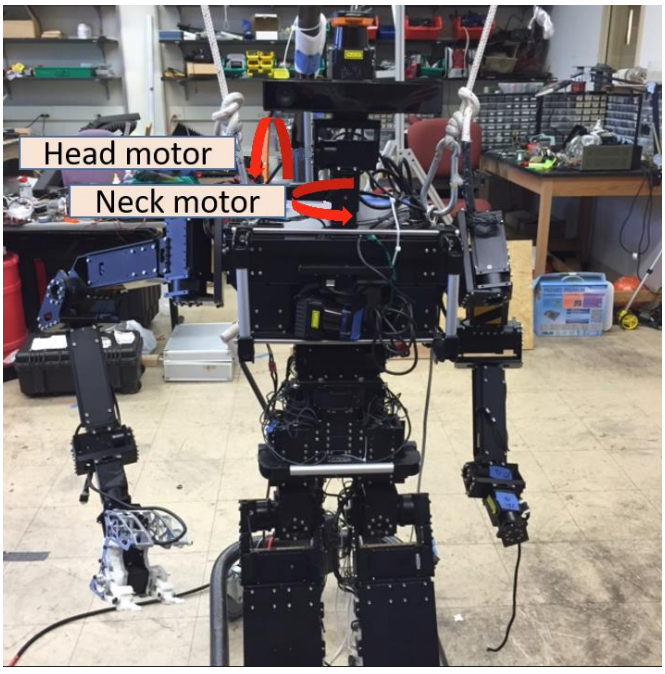
Project 3: Simultaneous Localization and Mapping

# Introduction:

Simultaneous Localization and Mapping (SLAM) is a topic of study that has been around for many years, but has gained more and more interest as of late. SLAM creates a map of an environment either unknown or known, and it keeps track of where an agent has been and where they are estimated to be inside of the generated map. Due to increasing interests in robotics, automation, unmanned vehicles and self-driving cars, SLAM is more important now than ever. One of the main aspects of self-driving cars is SLAM itself, self-driving cars have to be able to map out the environment around them in order to know where they are in relation to other cars and other obstacles. SLAM can be implemented with various different algorithms and filters, and can be implemented in 2D or 3D. This makes SLAM very versatile for any system that requires robotics or automation.

# Problem Formulation:

In this project we are creating a SLAM system based around the robot THOR –



This robot is equipped with –

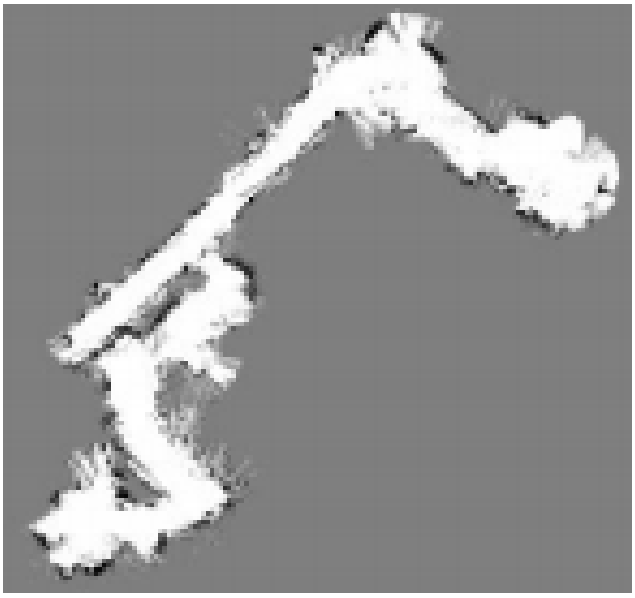
- ArdulMU+V2 Inertial Measurement Unit
- Hokuyo LIDAR sensor
- Kinect v2 sensor
- And head and neck motors

This allows the robot to record accelerometer and gyroscope values, LIDAR readings in a 360 degree range around the robot, as well as camera images of the surroundings from the Kinect sensor. Using this data we want to find a way to map this robot in our environment. This data gives us enough information to determine how the robot is moving, and what it is detecting.

Using this information we want to find an initial starting point for our robot, then map out the environment around it. Once we know the environment we can estimate where the robot is likely to move in the surrounding area. This will be the prediction step of our filter. After we predict where the robot will move to, we can use an update step of our filter to update our predictions based on the new data for the next time step that we evaluate. Then we modify our previous predications based on the update step, and repeat this process over all time steps that we have data for. Once this is done and we have our map generated, as well as the path of the robot through the map, we can use the data from the RGBD Kinect sensor to color the map.

## Technical Approach:

The idea is that we want to first create an occupancy grid where based on the LIDAR readings and the pose of the robot, we can create an initial map of the environment. This will be created solely from the sensor data, and will not take into account how the robot moves through this environment. Once the occupancy grid is created it will look like –



To start with we need to remove invalid readings, whether they be too close (<.1) or too far (>30), this will help to remove outliers and make the map more accurate. Once we remove the readings we can convert these from polar to Cartesian to create the vector –

$$(u_i, v_i, 0), i = 1....N$$

Then with this vector we need to convert from LIDAR to body, and then from body to world –

$$(w^T b)(b^T l) \begin{pmatrix} u_i \\ v_i \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} a_i \\ b_i \\ c_i \\ 1 \end{pmatrix}$$

The new vector on the right in terms of a, b and c represents the points that the LIDAR hit, where there is no obstacle along the ray to this point. Using these points, we increase the log odds of the cells that the LIDAR hit, and decrease the log odds of the cells that the LIDAR passed through. Once we make an occupancy map out of all of these log odds we get the initial map shown above.

Once we have the map generated we can now create the prediction step of the particle filter, to predict where our robot will move to. Initially we create N particles with the distribution –

$$u_{t|t} = (0, 0, 0)^T \epsilon SE(2))$$

Next we generate some noise to add in, from the model –

$$\omega_{t,i} \sim N(0, \sigma^2 I_3)$$

Then we use the odometry motion model along with the noise to compute the updated pose –

$$u^{(i)}_{t+1|t}, i = 1, ..., N$$

Once we do this for all of the particles (N), we can plot all of these back onto our map to get the new map after the prediction step of the particle filter.

Now we need to update this map's particle positions and weights based on the next time step of data. We can calculate the following, using the GetMapCorrelation function (corr) –

$$P(scan|x_{t,i}, \lambda_t) = exp(corr(scan, \lambda_t))$$

Then converting from LIDAR to body and body to world like before, we can get the values where $\lambda_t$ is the log odds of the map, and scan is –

$$scan = \begin{pmatrix} a_i \\ b_i \\ c_i \\ 1 \end{pmatrix}$$

Now again like before we can remove points that are too close or too far away. Next we can compute –

$$corr(m_t, z_t^{world})$$

Where –

$$corr = c_i, i = 1, ..., N$$

Finally using these correlation values we can compute –

$$\alpha_{t+1,i} = \frac{\alpha_{t,i} * e^{c_{t,i}}}{\sum_{j=1}^{N} \alpha_{t,i} * e^{c_{t,i}}}$$

After which we find which particle has the largest weight out of all N particles, choose that as our current point, and repeat the prediction to update steps as shown above. Repeating this process until all data points are evaluated. This results in a SLAM system that has the pseudo code process –

1. Find particle $\mu_{t|t}^{*}$ with highest weight
2. $\mu_{t+1|t}^{(i)} \leftarrow LocalizationPrediction(\mu_{t|t}^{(i)}, o_t, o_{t+1})$
3. $(\mu_{t+1|t+1}^{(i)}, \alpha_{t+1|t+1}^{(i)}) \leftarrow LocalizationUpdate(\mu_{t+1|t}^{(i)}, \alpha_{t|t}^{(i)}, z_{t+1}, m_t, {}_bT_h)$
4. $m_{t+1} \leftarrow Mapping(z_{t+1}, \mu_{t|t}^{*}, {}_bT_h, m_t)$

Finally once we have our map generated we can proceed with texture mapping to color it. Using the robot's pose we want to transform our RGB image ($I_t$) and our depth image ($d_t$) via RGB to body and IR to body transformations. Using the RANSAC method we can find the ground plane of the transformed data, from which we color the cells of the texture map based on the RGB values of $I_t$ and $d_t$ that are ground plane.

Once this is all done we now have a colored map that shows the location of obstacles that the robot has detected, as well as the starting point, path, and ending point of our robot as it moves through this environment.