James Madison University

CS 497: Autonomous Vehicles

Team: Artificial Intelligence/Robotics

Sprint #5 Report

1. Abstract

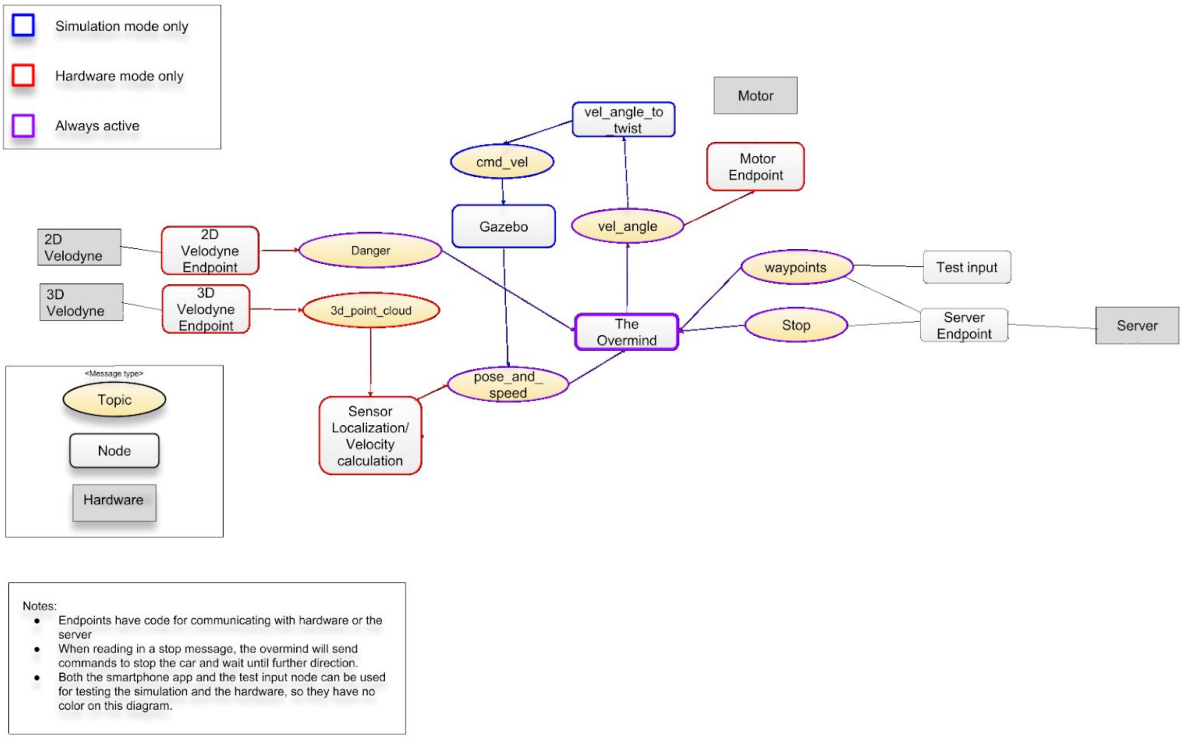
The Robotics team of the Autonomous Vehicles course was in charge of updating and improving the autonomous navigation of the golf cart on the software side. The core of the system used ROS (Robotic Operating System) and Autoware an open source autonomous driving software that we used for localization. We introduced solutions to issues such lidar localization integration with navigation, reduced complexity of launching navigation, improved documentation, etc.

2. Goals

Here are the goals we aimed to complete and the current status:

1. Localization With Autoware: Using Lidar and Autoware we are able to localize our cart within a point cloud map that will allow us to properly navigate
2. Proper Navigation and Localization in our program: After Autoware was providing localization we had to use the information it provided through an array of poses in order to determine our location.
3. Motor Endpoint Ranges: By using the sample bag file we ran our navigation at the same time and measured the full range of steering and meaning of velocity and velocity current in order to properly inform and interact with the cart's hardware
4. Integration with Actual Cart Hardware: Currently waiting for electronics team to get the cart operational
5. Fix GPS to XYZ Translation System: This system is currently functioning but only because it has been manually configured for the current point cloud map.
6. Obstacle Avoidance: This functionality would help to avoid issues with objects in the road from grabage cans, people, cars, etc. It is currently a work in progress and may not be working in time for the dry run
7. Bus Route Navigation: Using a set of predetermined waypoints we build a directed graph that is then used to create a global plan based on current location and target location.

3. **Hardware:**
    a. **Velodyne LIDAR**
        i. **The LIDAR we used with this cart is supported through a set of libraries in ROS and was the same LIDAR used by the Autonomous Vehicle team from previous years. As a result this means integration of this hardware into our program has been mostly straightforward and we hope it will work as intended**
    b. **2D LIDAR (details TBA)**
    c.
4. **Architecture Overview**



Above is the overview of our architecture. The rounded rectangles represent our nodes, where the processing of data and decision making takes place. The ovals represent topics, which is where data is sent. Lastly the squares represent the physical hardware that we interact with. This part of the documentation will be expanded later.

5. Mind Python File

Our Mind Python File is the core node that brings all the other nodes together to create our navigation system.

```
#!/usr/bin/env python

import math
import gps_util
import geometry_util
import rospy
from navigation_msgs.msg import WaypointsArray, VelAngle
from nav_msgs.msg import Path, Odometry
from std_msgs.msg import Header, Float32
from geometry_msgs.msg import PoseStamped, Point, TwistStamped, Pose, Twist
from visualization_msgs.msg import Marker
import tf.transformations as tf

import cubic_spline_planner
import pure_pursuit

'''
This class contains the code to track and fit a path
for the cart to follow
'''
class Mind(object):
    def __init__(self):
        rospy.init_node('Mind')

        self.odom = Odometry()
        self.gTwist = Twist()
        self.gPose = Pose();
        self.google_points = []
        self.rp_dist = 99999999999
        self.stop_thresh = 5 #this is how many seconds an object is away

        #waypoints are points coming in from the map
        self.waypoints_s = rospy.Subscriber('/waypoints', WaypointsArray,
                                            self.waypoints_callback, queue_size=10)
        self.odom_sub = rospy.Subscriber('/pose_and_speed', Odometry,
                                         self.odom_callback, queue_size=10)
        self.rp_distance_sub = rospy.Subscriber('/rp_distance', Float32,
                                                self.rp_callback, queue_size=10)
        self.twist_sub = rospy.Subscriber('/estimate_twist', TwistStamped, self.twist_callback, queue_size = 10)
        self.pose_sub = rospy.Subscriber('/ndt_pose', PoseStamped, self.pose_callback, queue_size = 10)
        #publishes points that are now in gps coordinates
        self.points_pub = rospy.Publisher('/points', Path, queue_size=10, latch=True)
        self.path_pub = rospy.Publisher('/path', Path, queue_size=10, latch=True)
        self.motion_pub = rospy.Publisher('/nav_cmd', VelAngle, queue_size=10)
        self.target_pub = rospy.Publisher('/target_point', Marker, queue_size=10)
        self.target_twist_pub = rospy.Publisher('/target_twist', Marker, queue_size=10)

        rospy.spin()

    def odom_callback(self, msg):
```

Figure 1. Python code to start the Mind node and initialize the various topics that it must subscribe to and publish to.

This file was created by the previous team working in Fall 2018 and we have focused on improving it and understanding it in order to get navigation working properly again. This file as

fortunately been mainly the same as the additions have been made mostly in other additional nodes

6. Motor Endpoint

The motor endpoint node is in charge of receiving navigation messages and sending them over a serial port to the Arduino which will then communicate with the hardware and actually move the cart.

```
        rospy.init_node('motor_endpoint')
        rospy.loginfo("Starting motor node!")
        #Connect to arduino for sending speed
        try:
            self.speed_ser = serial.Serial(SPEED_PORT, 19200, write_timeout=0)
        except Exception as e:
            print "Motor_endpoint: " + str(e)
            rospy.logerr("Motor_endpoint: " + str(e))
            #exit(0)

        rospy.loginfo("Speed serial established")
        """
        Connect to arduino for steering
        """
        self.speed_string = ''
        self.motion_subscriber = rospy.Subscriber('/nav_cmd', VelAngle, self.motion_callback,
                                                  queue_size=10)
        self.killswitch_subscriber = rospy.Subscriber('/emergency_stop', EmergencyStop,
                                                      self.kill_callback, queue_size=10)

        rate = rospy.Rate(5)

        while not rospy.is_shutdown():
            if self.cmd_msg is not None:
                self.send_to_motors()
            rate.sleep()

    def motion_callback(self, planned_vel_angle):

        if self.killswitch:
            self.speed_ser.write(":0.0,0.0,0.0".encode())
            rospy.loginfo("Killswitch activated")

        self.cmd_msg = planned_vel_angle


    def kill_callback(self, data):
        self.killswitch = data.emergency_stop


    def send_to_motors(self):
        target_speed = self.cmd_msg.vel #float64
        current_speed = self.cmd_msg.vel_curr #float64
        target_angle = self.cmd_msg.angle #float64

        data = (target_speed,current_speed,target_angle)
        data = bytearray(b'\x00' * 6)
        rospy.loginfo(data)
        bitstruct.pack_into('u8u8u8u8', data, 0, 42, 21,
                            target_speed, current_speed, target_angle)
        self.speed_ser.write(data)
        #rospy.loginfo(data)
```

Figure 1. Python code to take in navigation commands and send data to the arduino hardware

The function send_to_motors had to be analyzed and changed in order to adjust to the new hardware compared to the older cart that the original creator had in mind. We have updated this function to be more straightforward and have record the ranges of both the speed and angle so that we can properly coordinate with the electronics team. The motor endpoint uses a topic called emergency_stop to handle stopping quickly, the reason it is designed this way is so that at anytime any node setup to do so can send a message to the motor endpoint so that it knows to stop as quickly as possible, this message would then be passed on to the cart itself.

7. Landmarks to Waypoints Node

This node was created to demonstrate the functionality of pre-planned waypoints for how to get from one location to another. This node takes in a request from the networking team in the form of “currentlocation_targetlocation” this node may be obsolete if we decide to dynamically create the waypoints.

```
#!/usr/bin/env python

import rospy
import os
from navigation_msgs.msg import WaypointsArray, LatLongPoint, Landmarks
from sensor_msgs.msg import NavSatFix

class landmarks_to_waypoints(object):
    def __init__(self):
        rospy.init_node('landmarks_to_waypoints')

        self.waypoint_pub = rospy.Publisher('/waypoints', WaypointsArray, queue_size=10, latch=True)
        self.landmark_sub = rospy.Subscriber('/landmarks', Landmarks, self.landmark_callback, queue_size=10)

        rospy.spin()

    def landmark_callback(self, msg):
        landmarks_waypoints = msg.landmarks
        f = open (os.path.dirname(__file__)+"/"+landmarks_waypoints.data+".txt", "r")
        p_array = []

        for line in f:
            l = NavSatFix()
            items = line.split(",")
            l.latitude =  (float) (items[0])
            l.longitude = (float) (items[1])
            l.altitude = (float) (items[2].split("\n")[0])

            p_array.append(l)


        msg = WaypointsArray()
        msg.waypoints = p_array
        self.waypoint_pub.publish(msg)
        print "Waypoints added for: " + msg.landmarks.data

if __name__ == "__main__":
    try:
            landmarks_to_waypoints()
    except rospy.ROSInterruptException:
            pass
```

Figure 1. Python code to take in a String and open the associated waypoints file

Autoware Localization

We are using Autoware’s NDT_Matching localization algorithm to localize the cart’s position on a given point cloud map. NDT Matching is a relatively low resolution localization algorithm which is a fair tradeoff for the performance it has. The NDT_Matching algorithm publishes a

Pose message which consists of an X, Y, and Z of the local coordinates of where the localizer thinks the cart is on the map. This message comes from the topic, “ndt_pose”. NDT_Matching also gives a Twist message of what it believes the current velocity of the cart is, this is given to us under the topic, “estimate_twist”. These two messages give us fairly accurate positioning and speed of the cart in real time.

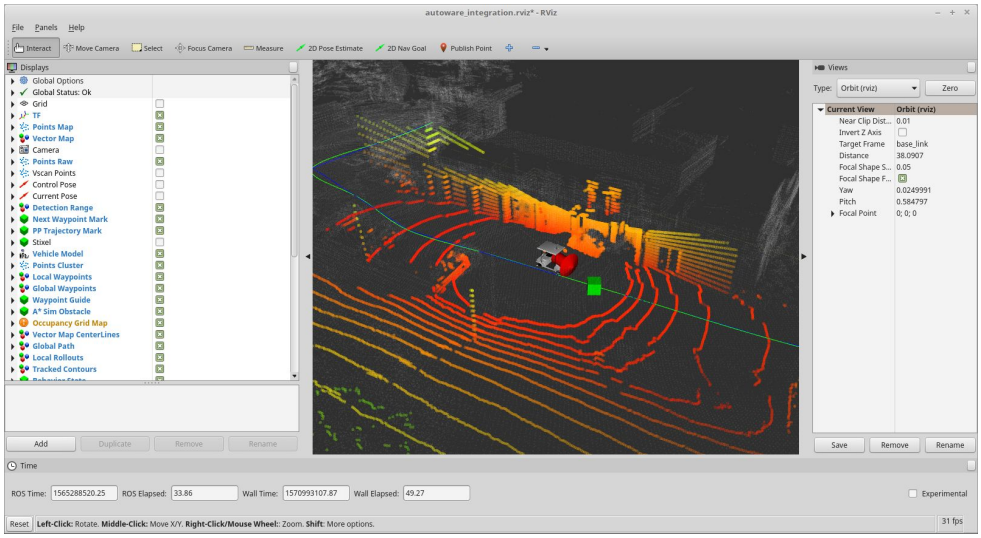


Figure 1. Autoware Localization with Map

8. Steering

Our team worked on finding the ranges of the angle for the steering wheel. In order to give it to the Hardware & Monitoring team. We started testing using the simulation of the golf cart along with teleop.py. We were able to find the range of the teleop was from -70 going left and 70 going right maximum. In addition, we used the rostopic on nav_cmd to listen on the topic to see the current angle at that moment of time. We looked through the output from the topic to figure out the ranges as well. For further research on the steering angle we ran used the same topic to look through the current angle while we ran the navigation localization code with it. This allowed us to find that the range was somewhere between (-58,59) with 0 being the neutral, meaning going straight forward with no curve.

The following commands we used to find the range of the steering angle are:

First Run.

```
roslaunch cart_control navigation.launch testing:=1
rostopic echo nav_cmd
rosrun cart_simulator teleop.py
```

Second Run.

roslaunch cart_control navigation_localization.launch
rostopic echo nav_cmd
rosrun cart_simulator add_test_points.py
rosbag play --clock OnlyPoints.bag

Smart Bus Route Design:

Using predetermined waypoints the cart can navigate from a start location to a target location. It will also make the correct decision at the intersection of XLABS for the shortest route. This logic is predetermined as we found that regardless of how we designed it our current navigation system would require hard coding the decisions in some way. Though we are prototyping two different ways of storing these waypoints and how to make the path. This design should be ready for testing for the Nov. 11 dry run.