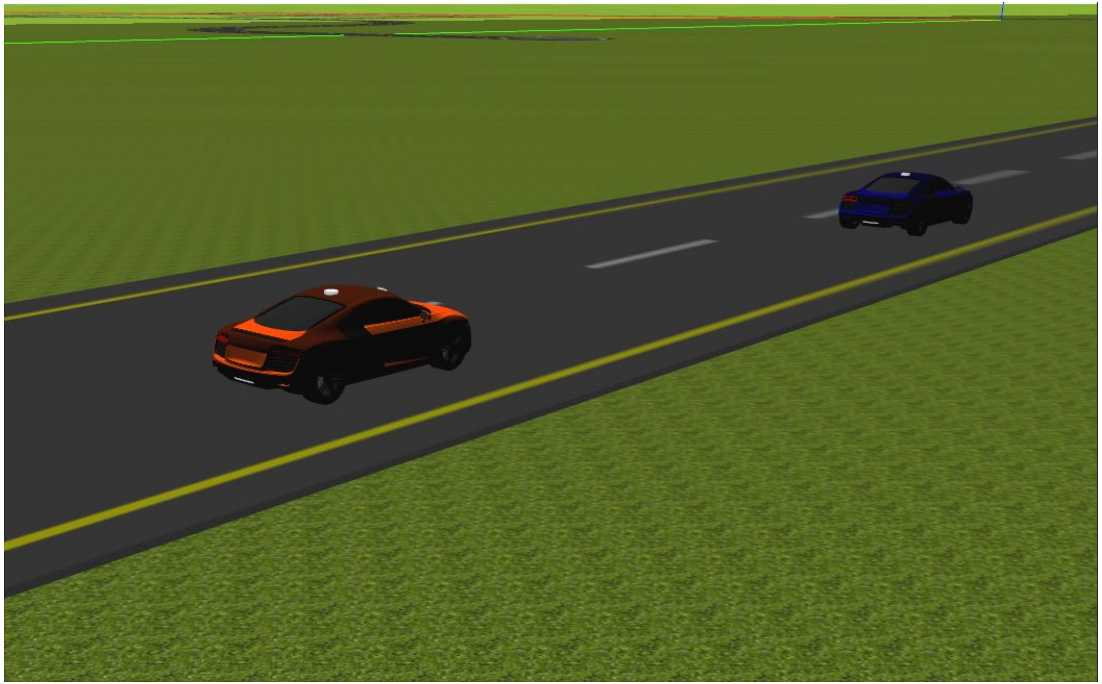
Audibot Adaptive Cruise Control



ECE 5532 - Winter 2021

April 26th, 2021

Team Members:
Lisa Branchick - BSME
Aaron Garofalo - BSME
Brian Neumeyer - BSME

Prepared for:
ECE 5532
Winter 2021
Dr. W. Geoffrey Louie

Group Members

Brian Neumeyer

- Bachelors in Mechanical Engineering
- Engineering Lead- Geofabrica

Aaron Garofalo

- Bachelors in Mechanical Engineering
- Development Engineer - MAHLE

Lisa Branchick

- Bachelors in Mechanical Engineering
- Embedded Controls Engineer - Ford

Project Introduction

“Adaptive cruise control layers convenience onto non-adaptive systems, which can maintain a desired speed on the highway.

...Adaptive cruise control takes that convenience a step further by allowing the driver to set a desired speed and following distance from any vehicle that may be ahead. If a slower vehicle moves in front of you, the system will automatically slow to maintain your pre-set following distance and then accelerate again to your originally set speed once the vehicle moves out of the way.” - MotorTrend

https://www.motortrend.com/news/adaptive-cruise-control/

Project Introduction

Three methods presented for Adaptive Cruise Control in ROS:

- Level 1: Vehicle Position
- Level 2: LIDAR
- Level 3: Camera

Level 1: Vehicle Position (GPS)

- Created a subscriber for /gazebo/model_states
    - “Look up the states of the two cars’ Gazebo models, extract their positions, and compute the distance between them”
- Within function recvModelStates() find A1 & A2 objects in ModelState array (always last two elements)
- Extract locations from ModelStates

```
void recvModelStates(const gazebo_msgs::ModelStates& msg){
  int arraySize = msg.name.size();    //vehicle models are
  //last in array is lead car
  //second last is follow car
  double a1x = msg.pose[arraySize-2].position.x;
  double a1y = msg.pose[arraySize-2].position.y;
  double a1z = msg.pose[arraySize-2].position.z;

  double a2x = msg.pose[arraySize-1].position.x;
  double a2y = msg.pose[arraySize-1].position.y;
  double a2z = msg.pose[arraySize-1].position.z;

  a1_a2_separation = cartDistance(a1x, a2x, a1y, a2y);

  pid_source = sep_target - a1_a2_separation;
```

Level 1: Vehicle Position (GPS)

```
//finds pythagorean distance between two xy points
double cartDistance(double x1, double x2, double y1, double y2)
{
  double xDiff = pow(x1-x2,2);
  double yDiff = pow(y1-y2, 2);

  return sqrt(xDiff+yDiff);
} //nice
```

- cartDistance() calculates distance between the two vehicles
  - Pythagorean distance
  - Based on Gazebo ModelStates locations of the vehicles
- Returned value is compared to target distance
  - Difference between target distance and current distance is input to PID
  - Output to /a1/cmd_vel using geometry_msgs::Twist

Level 2: LIDAR

```xml
  <arg name="urdf_file" if="$(eval camera and gps and lidar)" value="audibot_gps_and_camera_and_lidar" />
  <arg name="urdf_file" if="$(eval camera and gps and not lidar)" value="audibot_gps_and_camera" />
  <arg name="urdf_file" if="$(eval camera and not gps and not lidar)" value="audibot_camera" />
  <arg name="urdf_file" if="$(eval gps and not camera and not lidar)" value="audibot_gps" />
  <arg name="urdf_file" if="$(eval not gps and not camera and not lidar)" value="audibot" />

  <arg name="urdf_gps_args" if=    "$(arg gps)" value="gps_rate:=$(arg gps_rate) ref_lat:=$(arg ref_lat) ref_lon:=$(arg ref_lon)" />
  <arg name="urdf_gps_args" unless="$(arg gps)" value="" />

  <arg name="blue_arg" if=    "$(eval color=='blue')" value="blue:=true" />
  <arg name="blue_arg" unless="$(eval color=='blue')" value="blue:=false" />

  <group ns="$(arg robot_name)" >
    <param name="robot_description" command="$(find xacro)/xacro '$(find ece_5532_final)/urdf/$(arg urdf_file).urdf.xacro' pub_tf:=$(arg pub_tf) robot_name:=$(arg robot_name) $(arg urdf_gps_args) $
    <node pkg="gazebo_ros" type="spawn_model"  name="spawn_$(arg robot_name)" args="-urdf -param robot_description -model $(arg robot_name) -x $(arg start_x) -y $(arg start_y) -z $(arg start_z) -Y
    <node pkg="robot_state_publisher" type="robot_state_publisher" name="state_publisher">
      <param name="publish_frequency" type="double" value="$(arg tf_freq)" />
      <param name="tf_prefix" value="$(arg robot_name)" />
    </node>
  </group>

</launch>
```

```xml
<?xml version="1.0"?>
<robot name="audibot_mod" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <xacro:include filename="$(find ugv_course_sensor_description)/urdf/hokuyo_utm_30.urdf.xacro"/>

  <xacro:hokuyo_utm_30 name="laser_front" parent_frame="base_footprint" x="3.5" y="0.0" z="0.75" roll="0.0" pitch="0.0" yaw="0.0" />

</robot>
```

URDF files associated with level_2.launch were updated to include the xacro for LIDAR sensor

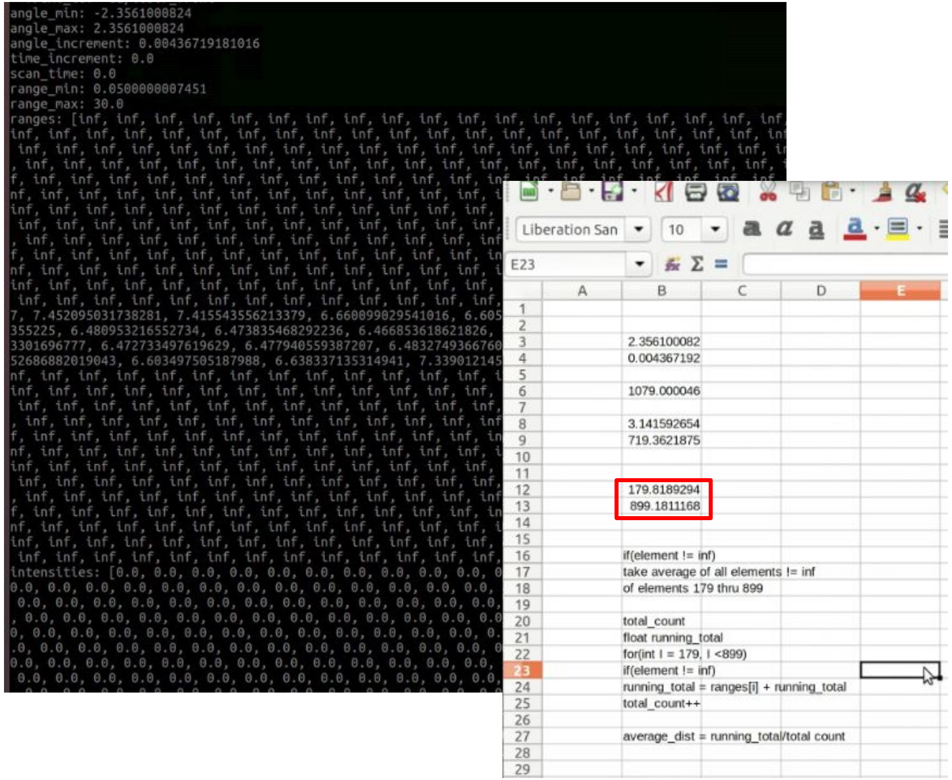
Level 2: LIDAR

```cpp
int main(int argc, char** argv){
    ros::init(argc, argv, "lidar_nav");
    ros::NodeHandle nh;

    //initializing PID object, passes pointer
    PIDController<double> *ip = &vel_PID_controller;
    initPID(*ip);

    //timer to refresh PID
    ros::Timer PID_timer = nh.createTimer(ros::Duration(0.01), PIDTimerCallback);

    //for publishing steering and throttle messages
    pub_vel = nh.advertise<geometry_msgs::Twist>("/a1/cmd_vel", 1);

    ros::Subscriber sub_cmd_vel = nh.subscribe("/a1/cmd_vel", 1, recvThr);

    ros::Subscriber sub_lidar = nh.subscribe("a1/laser_front/scan", 1, recieveLaserScan);

    ros::spin();
}
```

- Main() calls subscriber for LIDAR point cloud
- Information obtained from functions is processed by PID
- Vehicle velocity is then published using geometry_msgs

Level 2: LIDAR

- “rosmsgs show sensor msgs/LaserScan”
- Number of values calculated using angle_max and angle_increment
- Range reduced to front of vehicle
- Ignore values that are “inf” or “0.0”



Level 2: LIDAR

- Create func. recieveLaserScan()
- For the array “ranges” loop through values 179 to 900
- If value is in the valid range, add it to total_run and increment total_count
- Find the average of the ranges
- Create if statement to keep the vehicle at cont. speed if out of range
    - LIDAR max sensor range is 30m
- Use separation value in PID

```
void recieveLaserScan(const sensor_msgs::LaserScan::ConstPtr& msg){
//store collected messaged in array
  float run_total = 0.0;
  int total_count = 0;
  //angle_max*angle_increment = number in array
  for (int i = 179; i < 900; i++){
    //avgerage all the elements in the array execpt inf
      if (msg->ranges[i] < msg->range_max && msg->ranges[i] > msg->range_min){
        run_total += msg->ranges[i];
        total_count -=- 1; //for the memes
        ROS_INFO("msg range: %f", msg->ranges[i]);
      }
  }
  double avg = (double)(run_total/total_count);

  if(isnan(avg))
  {
    a1_a2_separation = 29.9;
  }
  else
  {
    a1_a2_separation = avg;
  }

  pid_source = sep_target - a1_a2_separation;
  //thank the Ballmer Peak
  ROS_INFO("avg: %f", avg);
  ROS_INFO("separation: %f", a1_a2_separation);
}
```

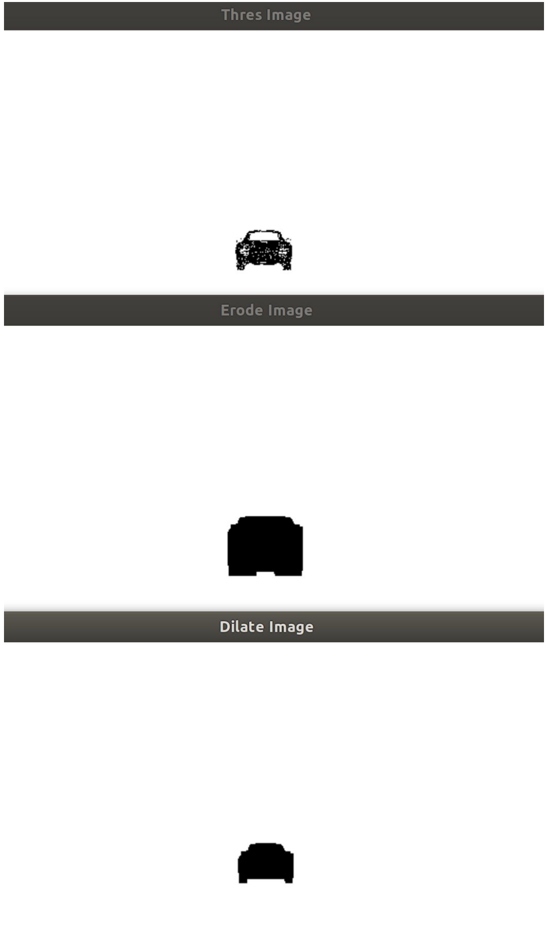
Level 3: Camera

- Used openCV_example as starting point
- Subscribed to /a1/front_camera/image_raw
- Extract blue channel from BRG original

```cpp
void recvImage(const sensor_msgs::ImageConstPtr& msg)
{
  int x_0 = 0;
  int y_0 = 0;
  int x_f = msg->width;
  int y_f = msg->width - 450; //cropping of vertical

  cv_bridge::CvImagePtr cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
  cv::Mat raw_img = cv_ptr->image;

  cv::imshow("Raw Image", raw_img);
  cv::waitKey(1);

  std::vector<cv::Mat> split_images;
  cv::split(raw_img, split_images);

  cv::Mat blue_image = split_images[1];
  //cv::Mat green_image = split_images[1];
  //cv::Mat red_image = split_images[2];

  cv::Mat croppedImage = blue_image(cv::Rect(x_0, y_0, x_f, y_f));
  cv::imshow("Cropped Image", croppedImage);

  cv::imshow("Blue Image", blue_image);
  cv::waitKey(1);

  cv::Mat thres_img;
  cv::threshold(croppedImage, thres_img, 2, 255, cv::THRESH_BINARY);

  cv::imshow("Thres Image", thres_img);
  cv::waitKey(1);
```

Level 3: Camera

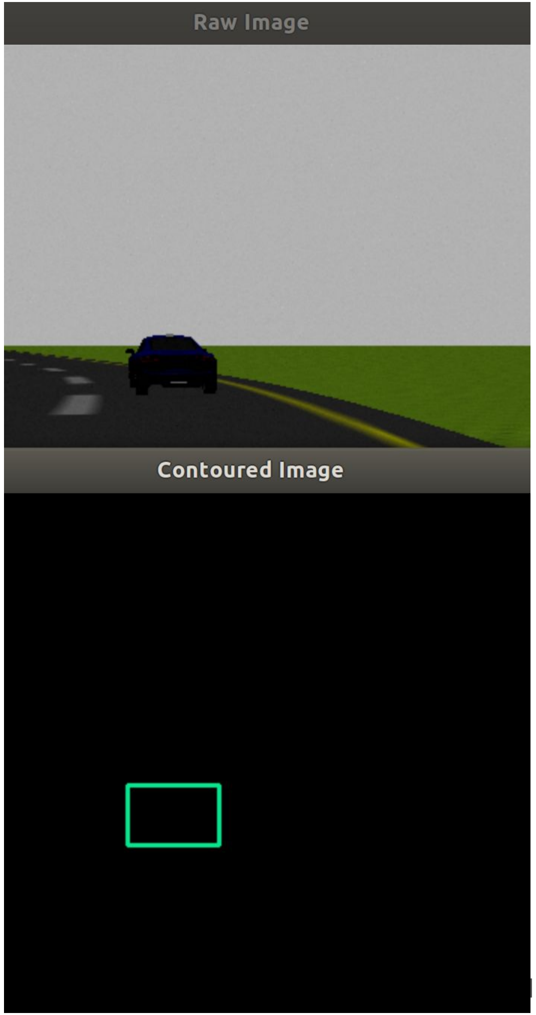
- Process image
    - Threshold pixels to show only mostly pure blue pixels
    - Erode and dilate image to make line classification more efficient
    - Draw contours and create bounding boxes
    - Tuning values were found using a dynamic reconfigure server to manually adjust



Level 3: Camera

- Extract vehicle height in pixels from bounding box
    - Height less affected by turning than width
- Pass pixel height to PID controller

```
    for( size_t i = 0; i < contours.size(); i++ )
    {
        approxPolyDP( contours[i], contours_poly[i], 3, true );
        boundRect[i] = boundingRect( contours_poly[i] );
        //minEnclosingCircle( contours_poly[i], centers[i], radius[i] );
    }
    Mat drawing = Mat::zeros( canny_output.size(), CV_8UC3 );
    for( size_t i = 0; i< contours.size(); i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
        //drawContours( drawing, contours_poly, (int)i, color );
        rectangle( drawing, boundRect[i].tl(), boundRect[i].br(), color, 2 );
        if(boundRect[i].height>max_height)
        {
          max_height = boundRect[i].height;
        }
        //circle( drawing, centers[i], (int)radius[i], color, 2 );
    }

    pix_height = max_height;

    //ROS_INFO("Number of contours: %d", contours.size());
    //ROS_INFO("max height: %d", pix_height);
    imshow( "Contoured Image", drawing );

    pid_source = pix_height - sep_target;

}
```



PID Controller

```
//PID constants
double P = 8.5;
double I = 0;
double D = 0;
```

```
//initializes PID components
void initPID(PIDController<double>& myDoublePIDControllerPtr){
  myDoublePIDControllerPtr.setTarget(pid_target);
  myDoublePIDControllerPtr.setOutputBounded(true);
  myDoublePIDControllerPtr.setOutputBounds(min_vel, max_vel);
  myDoublePIDControllerPtr.setEnabled(true);
}
```

- PID library was downloaded online - (https://github.com/nicholastmosher/PID)
  - Includes setup functions and pre-programmed controller
- initPID() sets up target deviation and min/max speed
  - Minimum speed ensure vehicle always moves forward
  - After values are set, PID is enabled
- Only P (proportional) was used
  - Maintains smooth vehicle operation
  - Reduces time and space complexity of the program
  - Future enhancement to implement full PID control for varying velocities and stability

PID Controller

```cpp
//refreshes PID
void PIDTimerCallback(const ros::TimerEvent& event){
  vel_PID_controller.tick();
  /*ROS_INFO("PID ticked");
  ROS_INFO("PID Target: %f", vel_PID_controller.getTarget());
  ROS_INFO("PID error: %f", vel_PID_controller.getError());
  ROS_INFO("PID output: %f", vel_PID_controller.getOutput());
  ROS_INFO("PID feedback: %f", vel_PID_controllegetFeedbackr.()); */
}
```

```cpp
//init PID controller
PIDController<double> vel_PID_controller(P, I, D, pidDoubleSource, pidDoubleOutput);
```

- ros::Timer is used to create timer callback for PID
    - 100 Hz frequency was used for smooth operation
    - Updating too slow can lead to undesirable vehicle behavior (drifting)
    - ROS_INFO was used to ensure variables updated properly
- Callback function triggers all PID functions to update
    - Updates input, calculation, and output
    - PID calculations occur within the PID library

PID Controller

```
//publishes velocity and steering command messages
void cmdVel(double v)
{
  geometry_msgs::Twist vel;

  vel.linear.x = v;
  vel.angular.z = cmd_turn;

  pub_vel.publish(vel);
  //ROS_INFO("Published Velocity: %f", vel.linear.x);
  //ROS_INFO("Distance between cars: %f", a1_a2_separation);
}
```

- cmdVel was borrowed from previous projects to control the vehicle
  - Output from PID is passed to linear.x to control vehicle throttle
  - Steering angle is passed through unchanged
    - Steering calculations occur in path_following package
  - New velocity and steering commands are published to vehicle
  - Output to /a1/cmd_vel using geometry_msgs::Twist

Demonstration

https://github.com/algarofa/ECE_5532_Final_Project

https://www.youtube.com/watch?v=dQw4w9WgXcQ

Thank you!