# Autonomous Golf Cart

ECEN 5070: Creative Component

Faculty Advisor: Dr. Weihua Sheng

Advisory Committee Members:

- Dr. He Bai
- Dr. Weili Zhang

Aditya Mulampally
A11816595

Table of Contents

Introduction ..... 2
About the golf cart ..... 2
Summary of my Work ..... 2
Methodology ..... 3
Components of the System ..... 4
Servo Motor for the Steering ..... 4
Motor I/O for PWM Mode ..... 5
Feedback – Potentiometer ..... 6
Steps followed for PID tuning ..... 7
Speed Control of the Golf Cart ..... 9
Brief Description of the pedal ..... 10
Digital Potentiometer ..... 11
Methods and Observations ..... 12
LIDAR ..... 13
Velodyne Lidar UDP data packet structure ..... 14
Point Cloud data ..... 14
ROS_Velodyne [9] [10] ..... 16
Summary ..... 16
Bibliography ..... 18
Appendix ..... 19
Demo Link ..... 19
Source Code: Servo Control ..... 19
Source Code: Speed Control ..... 23

# Introduction

Autonomous vehicles, from an engineering point of view is a very good example on the usage of technology to automate a task that is being done by humans since the introduction of cars. Autonomous cars are the next revolution in technology that can have a direct impact to our day to day lives, economy and environment. [1] It is evident that our society is preparing itself for accommodating driverless cars into our day to day lives with the National Transportation Safety Administration in early 2016 stated that under federal law computers can now be considered legal drivers [1]. The other benefits of this technology would be the potential decline in the loss of human lives due to traffic related accidents.

In this report I will be briefly discussing the efforts that I have taken to convert a commercially available electrical golf cart into a vehicle with autonomous driving capabilities. I will be first discussing about the components that were added to the golf cart to make it autonomous and will be discussing the challenges that were faced during the integration process.

# About the golf cart

The golf cart that is being worked upon is the *Club Car Precedent i2 Excel (2011)*.

- It is powered by six, 8-volt batteries that is used to run the electrical motor.
- It has an onboard computer that governs the charging of the golf cart with features such as auto-cutoff when full and it has a memory of the battery percentage since last charging in the case of an incomplete charging.
- It has a standard steering wheel as found on older automobiles.
- It has two gears, Forward mode and Backward mode.
- An accelerator and a brake pedal.
- The brake is a mechanical drum brake type on each of the rear wheels.

## Summary of my Work

- Establishing control and tuning of the PID loop on the servo motor mounted on the steering wheel.
- Establishing speed control of the golf cart via a digital potentiometer.
- Integration of the LIDAR as a ROS node.
- Creation of a ROS node for the speed and steering modules.

## Methodology

Making a vehicle autonomous involves adding electronics to emulate the functions that the human is performing to drive. These actions include the following:

- Steering input.
- Speed input.
- Awareness of the surroundings.
- Pristine sense of judgement.

As it is known the last function would be the most important for our day to day driving as it is directly responsible for the safety of the driver as well as the people around. To electronically emulate the above broadly defined functions the following actuators have been added to perform these functions by previous teams:

- A servo motor has been attached to the steering to give directional inputs.
- An electronic speed input module has been integrated.
- A LIDAR

In the following sections I will discuss the functionality of the above-mentioned integrations.

# Components of the System

## Servo Motor for the Steering

The servo motor being used for controlling the steering wheel to control direction is manufactured by Tecknic Inc. This motor is rated to operate under a voltage range of 24V-75V. It has a wide variety of modes it can operate under such as [2]:

- Spin on Power Up
- Manual Velocity Control
- Ramp Up/Down to selected Velocity
- Follow Digital Velocity
- Follow Digital Velocity (Bi-Polar PWM Input/ Frequency Input)
- Follow Digital Torque Command (Bi-Polar PWM Input/ Frequency Input)
- Move to absolute position
- Move Incremental Distance
- Pulse Burst Positioning

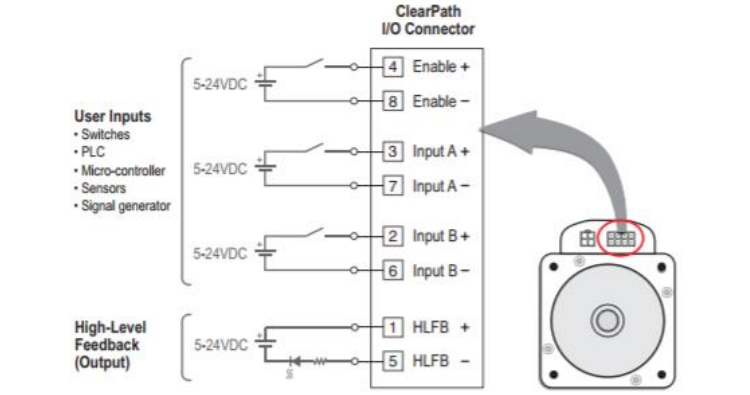
All these modes have been set by the manufacturer of the motor to enhance its functionality and versatility of the motor as opposed to a traditional servo motor that takes inputs from PWM inputs. The nameplate characteristics of this motor at 48V DC is as follows:

| | |
|---|---|
| Peak Torque | 9.6 N-m |
| RMS Torque | 3.4 N-m |
| Max Speed | 540 RPM |
| Peak Power | 148 W |
| RMS Power | 136 W |

As for a steering servo motor to respond to perfect angular inputs according to the requirement the BI-polar PWM mode was chosen where, positional commands can be given to the servo motor via PWM. To regulate and control the inputs that were given to the motor, a PWM loop was integrated to the system.

The following sections describe the input output pins of the motor and as well as the tuning of the PID control loop that was used to establish control of the motor.

## Motor I/O for PWM Mode




***Simplified overview of ClearPath inputs and output***

The above diagram represents the input and output ports that are used to command the motor in its selected mode. The following is the descriptions of each of its pins [2] [3] [4]:

**Enable Input** –

- Asserting the Enable input (logical 1, high, 5–24VDC) energizes the motor coils.
- De-asserting Enable (logical 0, low state, 0 volts) removes power from the motor coils.

**Inputs A and B** –

Once enabled, the servo motor can respond to the state of Inputs A and B.

- In this mode of operation
    - Input A Logic: High=Inhibit
    - Input A Logic: Low=Inhibit off
- Input B controls the direction and velocity according to the duty cycle given to the motor.

**High-Level Feedback (HLFB)** –

The servos HLFB output can be set up to alert the user or control system to one of several conditions. HLFB can be configured to:

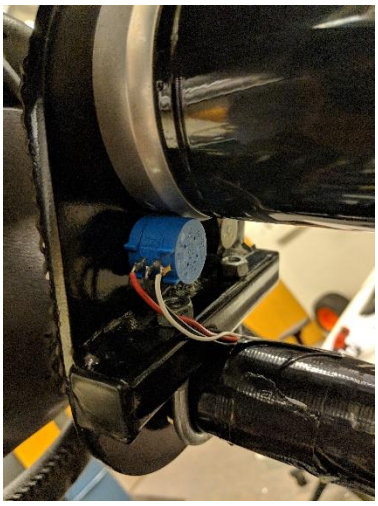
- Change state if a Shutdown occurs.
- Signal when the servo motor is running at your commanded velocity or torque.
- Signal the end of a move (based on user-defined settling requirements).
- Output a PWM signal whose duty cycle is proportional to motor speed, or torque.

As it can be seen form the HLFB outputs of the servo motor, it doesn't send out a clear description of the motor's current position to be clearly used for control purposes i.e., it doesn't have a feedback mechanism. Hence an external feedback mechanism must be installed to obtain this feedback data about its position.

## Feedback – Potentiometer

The knowledge of the position of the servo motor is important as we don't want to command the servo motor more than the steering can physically turn. For this purpose, a 5 K ohm multi-turn potentiometer is being used as the feedback mechanism for this motor. A potentiometer sends out various voltages when its knob is physically turned. The potentiometer used in this case can be turned for a maximum of 10 turns.

In this project the potentiometer is coupled to the servo motor via a gear mechanism such that when the motor turns the steering wheel, the potentiometer also turns along with it. The potentiometer is supplied a voltage of +5V from the microcontroller on its supply terminals and the output terminal is fed to the analog to digital converter of the microcontroller. This procedure converts the potentially dynamic changing voltages from the potentiometer to a 10-bit digital value that can be easily used to denote the position of the servo motor's rotation.



As seen in the pictures above, the potentiometer is coupled to the servo motor via gears, that can register the turns of the motor.

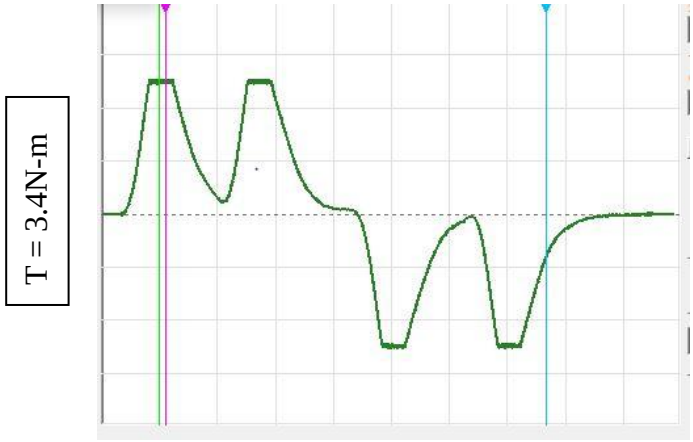
The wires can be described as follows:

- Red – Vcc
- Black – GND
- White – Output

## Steps followed for PID tuning

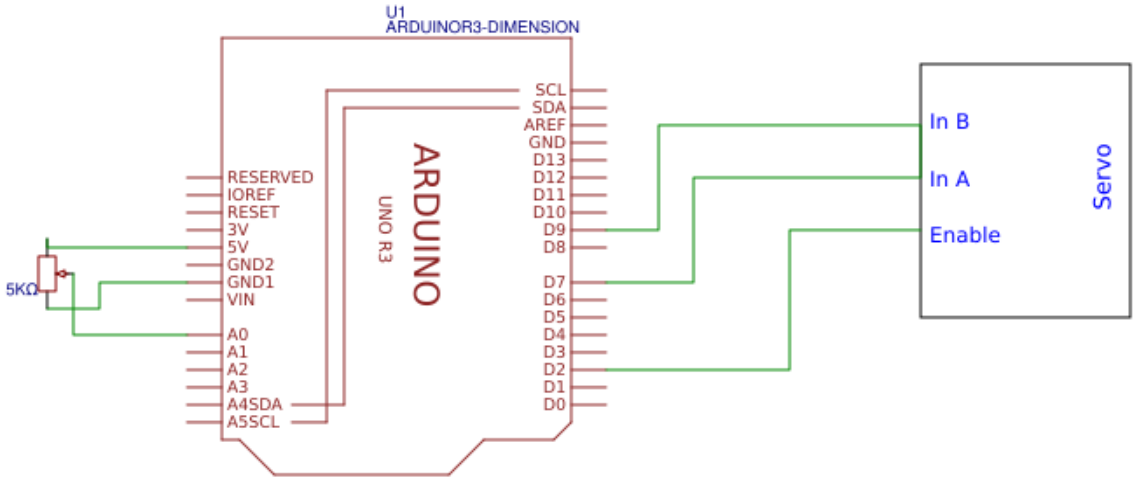
1. First a standard ADC was used to find out the extreme limits of the turns of the servo motor. For this step, the steering wheel was manually turned by hand and a note of the digital value was taken at each extremity of the turns i.e. left extreme and right extreme.
2. These values were used as absolute limits and care was taken via code that the steering never crosses such said limits.
3. The servo motor works in such a way that, when a PWM signal of less than 50% is applied, the motor shaft turns clock wise, when a PWN signal of greater than 50 % is applied, the motor turns counter clockwise. The maximum speed and the percentage of torque that is being applied to the system is set using the software on the PC. It is to be taken a note that this software is required only during setup and is not required afterwards.
4. There are two variables in the code that are the most important, the desired position and the actual position. The desired position is the final position to be achieved i.e. mid, left or right. The actual position is the position reported by the potentiometer.
5. A variable named error, is the difference between the actual and desired position. The main motive of the program is to reduce this error to ‘0’.

6. A PID controller was initially thought of being used to establish control of the system, with a manual approach of tuning the controller was followed [5].
7. Kp has been tuned first, with its value being slowly increased from 0 with a step of 0.1
8. The final value of Kp was set at 1, where there were uniform oscillations in the system, were it was decided to move onto to the tuning of the next parameter that is the Ki.
9. But unfortunately, any value of Ki caused the system to completely disobey my code, with the motor behaving abnormally, and completely spinning out of the bounds of safe operation.
10. Hence Ki was left at '0' for safety reasons. But the servo motor is accurate and there is no steady-state error present in the system, so I believe that only PD control is sufficient for the system. But this is just an assumption from my part and I don't have technical evidence to back it up.
11. Kd helped in reducing the settling times of the system. After incrementally increasing this parameter, the best value was found at 3.5
12. By tuning these both parameters Kp and Ki the system has optimal response and does not cause any abnormal oscillations around the set point.
13. The digital value for each corresponding degree has been calculated and this value has been used to give out heading in terms of degrees.




*The Torque vs Time Graph after PID tuning*

After the tuning process the following torque vs time graph that is obtained there are no overshoots or oscillations near the set points.




*Circuit Diagram for servo control*

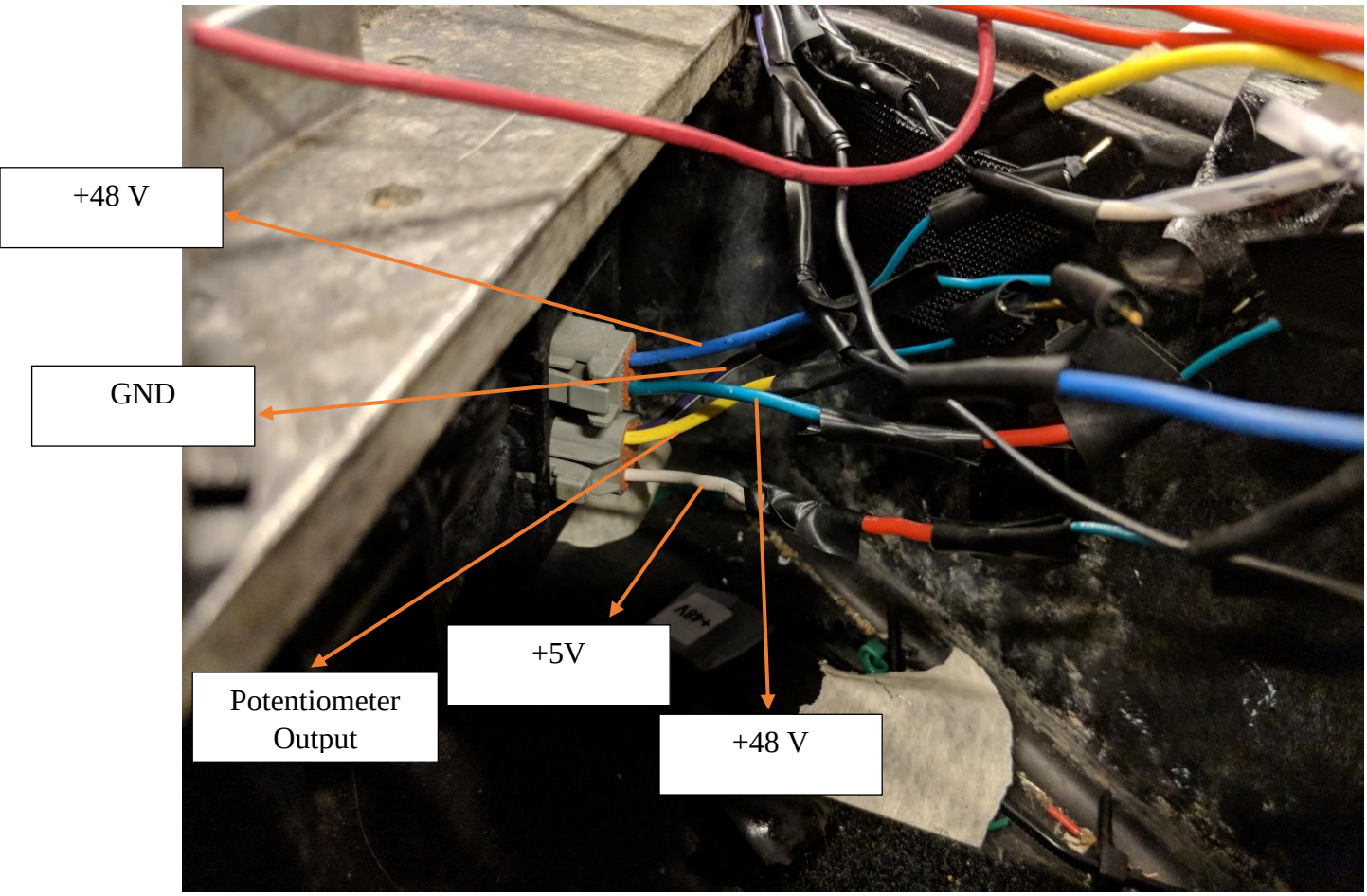
## Speed Control of the Golf Cart

The golf cart receives its speed commands form the accelerator pedal that sends the corresponding signal to the H-Bridge Motor controller to send power accordingly to the rear. I have found that the accelerator pedal is essentially a potentiometer that sends varied voltages according to the degree to which the accelerator is pressed.

The accelerator pedal reads:

- 0 volts when the accelerator is not being pressed.
- 5 volts when the accelerator is completely pressed.

## Brief Description of the pedal




- The pedal has 5 wires that come from it, as seen from the picture above and can be described as:
    - White - +5V for the high side of the potentiometer.
    - Purple-White – GND, the low side of the potentiometer.
    - Yellow- Output from the potentiometer.
    - Green, Blue – These wires both run 48V in them and must shorted together to signal that the inputs are coming from the pedal. (A safety feature)

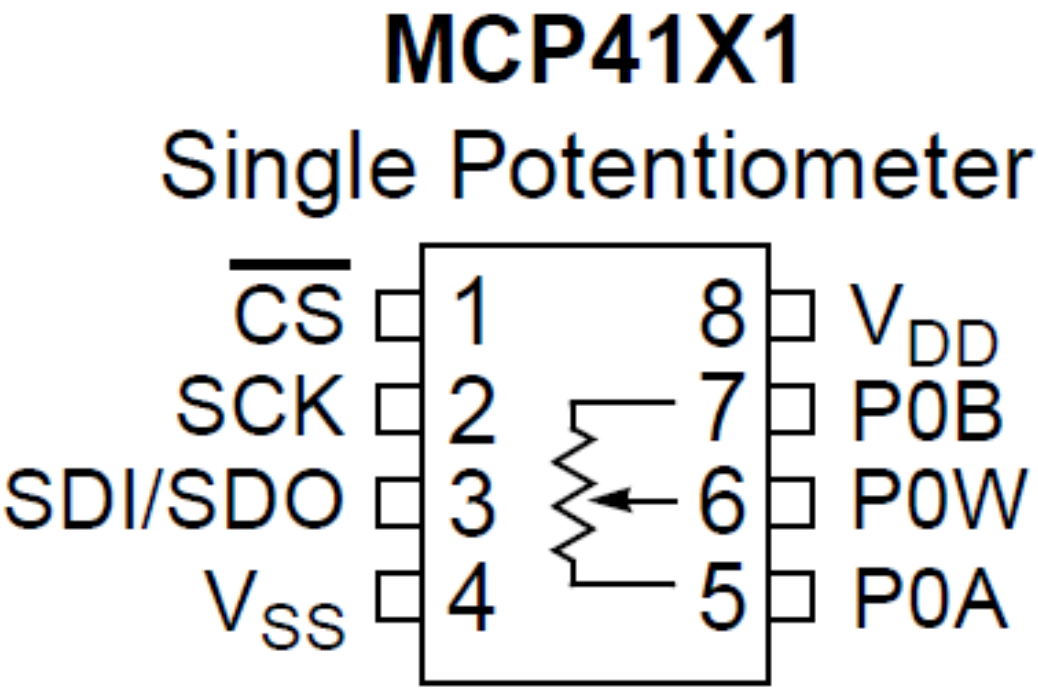
## Digital Potentiometer

To electronically emulate the actual physical pedal positions, we need a digital potentiometer that can be controlled via a microcontroller input. The digital potentiometer that is being used here is the MCP4131 by microchip [6].

The features of MCP4131 can be summarized below [7]:

- This potentiometer has a total resistance of 100KΩ.
- It operates between 1.8V – 5.5V
- Its network resolution is 7 bits i.e. it can vary the resistance in steps from 0-128.
- This gives us a resistance of about 775Ω per step.
- This digital potentiometer communicates with the microcontroller with the SPI protocol.

The PDIP map of this can be seen in the figure below:



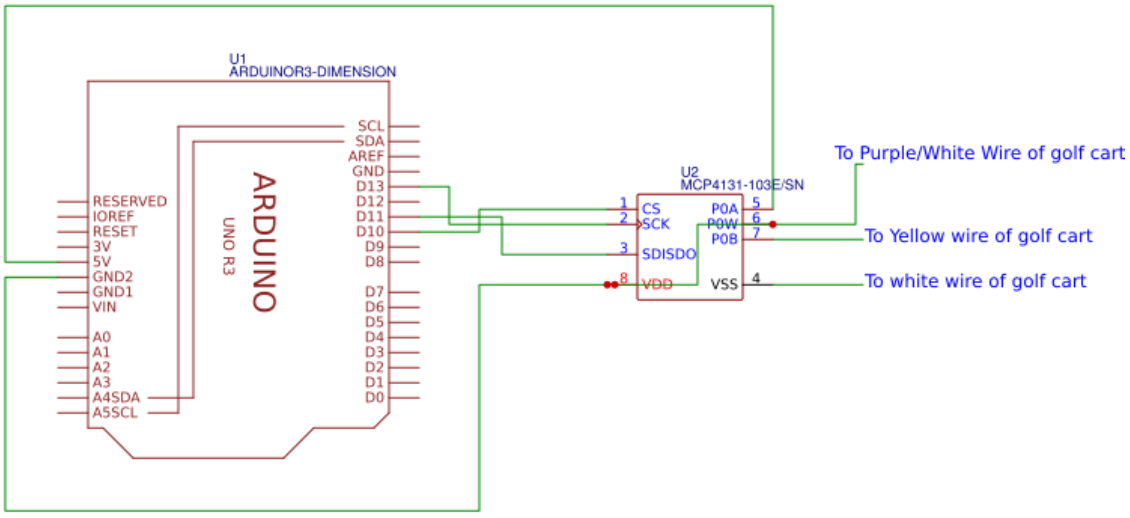
The detailed description of pin can be as summarized below [6]:

| Pin # | Pin Name | Description |
|---|---|---|
| 1 | CS | The CS, or Chip Select, pin is the SS (slave select) pin for the SPI interface. It is active low. A low means the chip is selected and high means it is not selected. |
| 2 | SCLK | SCLK is the Shared/Serial Clock. It is the SPI clock line. |
| 3 | SDI/SDO | These pins are the serial data in and out, also known as the MOSI and MISO. |
| 4 | VSS | This is where ground is connected to. |
| 5 | PA0 | This is one terminal of the potentiometer. |
| 6 | PW0 | This is the slider terminal of the potentiometer. |
| 7 | PB0 | This is one terminal of the potentiometer. |
| 8 | VCC | This is where the positive voltage source connects to. |

## Methods and Observations

1. It can be seen from the above table, that the chip must be powered up first before it can work as a potentiometer. The White and Purple/White wires, from the pedal assembly can be used for the power up of the chip.
2. The output of the digital potentiometer can be directly fed into the Yellow wire to directly command the H-Bridge controller for the speed of the rear wheels of the golf cart.
3. From my observation, the grounds of both the pedal assembly and the microcontroller are different, hence they should be attached together so that there is a common ground for the entire system.
4. The SPI was initiated from the microcontroller and the SPI values can vary from 0-128 as it is a 7-bit resistor network.
5. The wheels start spinning at a numerical value of 16 i.e. at a value of 12 KΩ i.e. 0.6 Volts.
6. The golf cart attains a maximum speed at 5 volts i.e. at a value of 128.
7. The H-bridge controller of the golf cart is responsible of regulating the speed input to the golf cart in terms of power.
8. As the motor is electric the change that can be attained in terms of torque and response time is very fast and can also be achieved through minimal wear and tear.
9. This apparatus of setup allows us to command very quick change in speeds and can be used in a case of emergency stoppage of the golf cart also.

The above sections cover the work related to the low-level control of the golf cart, the following section covers about the LIDAR, which is the long-range eyes of the cart.




*Circuit Diagram of the speed input module*

# LIDAR

LIDAR is an acronym for LIght Detection And Ranging. It is a very popular surveying tool used to measure to distance to the target from the source of the light. The light that is shone on the target can be of visible, ultraviolet or near infrared range of light.

For our golf cart a Velodyne 64 E S2.1 LIDAR is being used for detection of obstacles and to acquire information about long distant objects. One of the main advantages of utilizing a LIDAR sensor is its ability to analyze the distant data in a 360-degree field of view format in real time.

Salient features of the Velodyne 64E [8]:

| | |
|---|---|
| Range | 120m |
| Data points per second | 2.2 Million points per second |
| Horizontal FOV | 360 degrees |
| Vertical FOV | 26.9 degrees |
| Angular Resolution | 0.08 degrees |
| Vertical Resolution | 0.4 degrees |


The lidar has 2 laser blocks as seen in the figure, with 32 lasers on each block. The whole unit spins at a rate of 600 RPM which leads to roughly 1 million data points per second. The minimum distance that it can read is 3 feet and any readings that state a distance lower that this should be discarded.

## Velodyne Lidar UDP data packet structure

- The lidar sends its data via the ethernet port for the user to use the data.
- The entire size of the payload would be a size of 1206 bytes.
- The first two bytes indicate the source of the packet – upper or lower laser block.
- The next two bytes indicate the angle of rotation.
- The next bytes consist of 32 laser returns of size of 3 bytes each in an 8-bit intensity format, 0 being the weakest return and 255 being the strongest return.
- The last 6 bytes show the status of the lidar itself, which consists of information such as:
    - Internal temperature of the unit.
    - Firmware version number.

## Point Cloud data

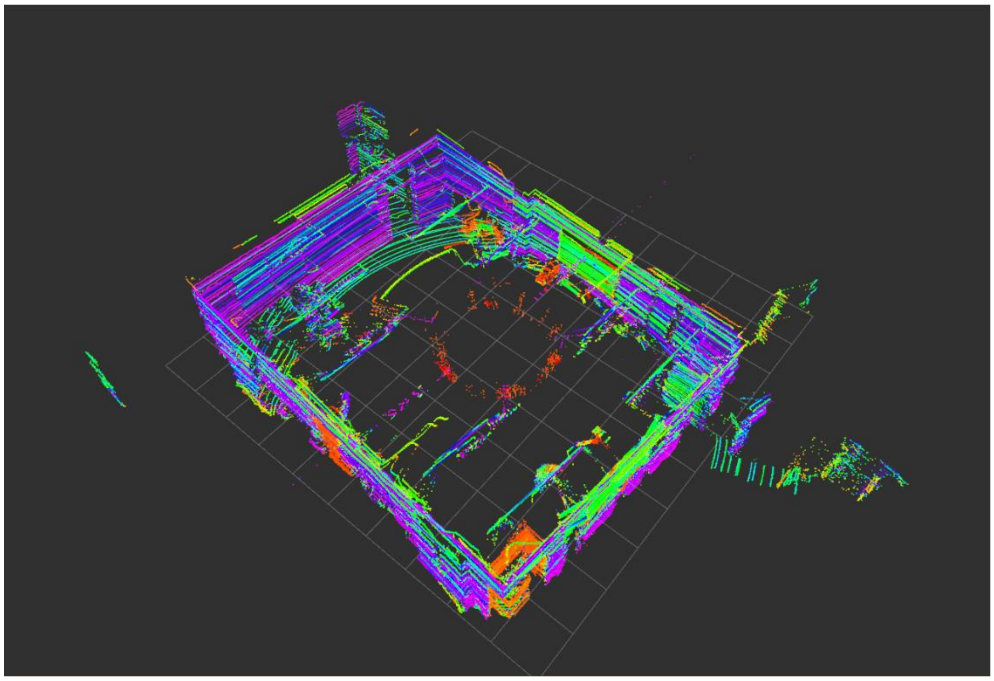
The data that is sent via these packets can be handled as Point Cloud Data. This point cloud data can be processed by the Point Cloud Library which is an opensource library to collect and process this data.

This point cloud data is of the sensor message – point cloud 2 data type format which contains the following descriptions:

- Width of the pointcloud.

- Height of the pointcloud.
- Length of the point data
- Actual point data

The information extracted from this can be obtained in ROS where its processing and visualization can be done of the point cloud.



The above image is the visual depiction of the data pointcloud data read in the lab.

The LIDAR is placed on top of the golf cart in this manner.

## ROS_Velodyne [9] [10]

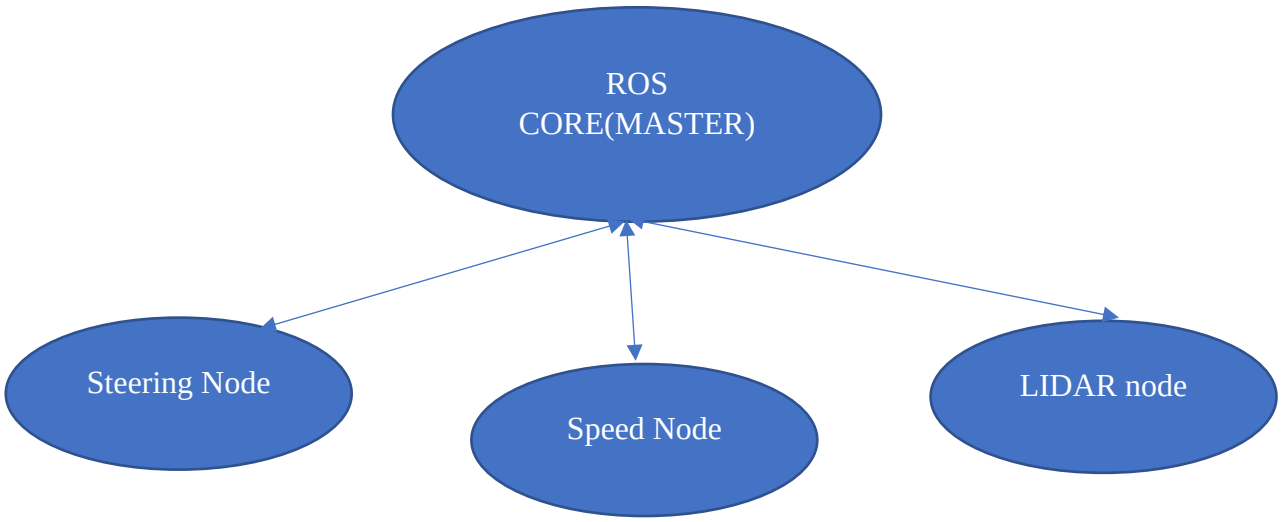
ROS is the chosen platform for its support for its various sensors, actuators and the various tools that it provides to process and visualize the data.

ROS has an official support package that is used to provide drivers to process and visualize the point cloud in rviz. This driver publishes the Velodyne packets node on the ROS network which is the packet data for 1 entire frame of reference. This data is available on the ROS network where this is used by the velodyne points node which can be used for complete visualization purposes in rviz.

The point cloud data that is obtained can be used to detect obstacles based on its height, by keeping a height threshold as done in the velodyne height map package [10].

## Summary

ROS is currently being used to integrate all the sensors and actuators on a common platform where they can communicate with each other and kept in check with the master software node. All the above stated works will become a separate node on the master ROS with the final architecture currently looking in this form:




This project is still currently ongoing with many other sensors yet to be integrated on to the ROS network such that they can communicate with each other to form a fully functional autonomous vehicle. There are other works such as brake control, GPS location node on ROS, and object

recognition via a camera that is to be integrated on the golf cart but they have not been described as they were handled by other researchers in the lab.

## Bibliography


[1] F. Dews, "Driverless cars are coming: Here are 8 useful facts about them," BROOKINGS NOW, 2017.

[2] "CPM-MCPV-3441S-RLS," [Online]. Available: https://www.teknic.com/model-info/CPM-MCPV-3441S-RLS/?model_voltage=75VDC.

[3] "Arduino - PWM," [Online]. Available: https://www.arduino.cc/en/Tutorial/PWM.

[4] "Clearpath Motors," 15 October 2017. [Online]. Available: https://www.teknic.com/files/downloads/clearpath_user_manual.pdf.

[5] "Understanding PID control," [Online]. Available: https://www.controleng.com/single-article/understanding-pid-control-and-loop-tuning-fundamentals/61530560e8cf5044c7291bda7e44b655.html.

[6] Microchip, "MCP4131 Datasheet," [Online]. Available: http://www.learningaboutelectronics.com/Datasheets/MCP-digital-potentiometer-ICs-datasheet.pdf.

[7] "How to Build a Digital Potentiometer Circuit Using a MCP4131," [Online]. Available: http://www.learningaboutelectronics.com/Articles/MCP4131-digital-potentiometer-circuit.php.

[8] "64 e user manual," [Online]. Available: http://www.velodynelidar.com/lidar/products/manual/HDL-64E%20Manual.pdf.

[9] J. O'Quin and J. Whitley, "velodyne," [Online]. Available: http://wiki.ros.org/velodyne?distro=kinetic.

[10] D. Claridge and M. Quinlan, "velodyne height map," [Online]. Available: http://wiki.ros.org/velodyne_height_map.

# Appendix

## Demo Link

https://photos.app.goo.gl/qJ6gJmuXNwMNiZPB7

## Source Code: Servo Control

```
/*
Name:    Servo_VS.ino
Created: 10/11/2017 6:10:39 PM
Author:  ADITYA
*/
#define mid  580                  //found by physically setting the steering
at mid
#define left_comp 695               //found by physically setting the
steering at left max
#define right_comp 463                //found by physically setting the
steering at right max
#define left 680
#define right 480
const int Enable = 2;               //for the servo to start working
const int InA = 7;                  //inhibit input has to be disabled for
the motor to spin
const int InB = 9;                 // the PWM input. 1 for direction 1 254
for direction 2
const int fdbk = A0;                //output of the encoder goes here
double currentpos = 0;                //variable for current position
double desired = mid;               //setting the desired position as mid
double pos_er = 0;                  //position error
double Outputval;

unsigned long lastTime= 0;
double errSUM, LastERR;
double sampletime = 0.1;
double Kp = 1, Ki = 0, Kd = 2.6;
double kp, ki, kd,a,b;

byte incdata;

void setup() {

  Serial.begin(9600);               //start serial comms
  pinMode(Enable, OUTPUT);            //setting enable, InA, InB in output
mode
  pinMode(InA, OUTPUT);
  pinMode(InB, OUTPUT);
  kp = Kp;
  ki = Ki * sampletime;
  kd = Kd / sampletime;
  a = kp * 1;
  b = kp * 100;
}

void loop() {
```

```
  currentpos = analogRead(fdbk);          //current postion will be the
direct input of the analog pin
  pos_er = desired - currentpos;          //self explanatory
//  Serial.print("CURRENT POSITION = ");      //Printing the needed variables
for observation
//  Serial.println(currentpos);
//  Serial.print("DESIRED POSITION = ");
//  Serial.println(desired);
//  Serial.print("ERROR TO BE COVERED = ");
  Serial.print(pos_er);
  Serial.print("\t");
  Serial.println("127");

  //delay(500);



  incdata = Serial.read();            //positional commands will be sent via
serial terminal

  if (incdata == 'q')
  {
    desired = left;
  }
  else if (incdata == 'a' && (currentpos < left_comp + 1))
  {
    desired = currentpos + 10;
  }
  else if (incdata == 'd' && (currentpos > right_comp - 1))
  {
    desired = currentpos - 10;
  }
  else if (incdata == 'e')
  {
    desired = right;
  }
  else if (incdata == 'w')
  {
    desired = mid;
  }


  while (pos_er != 0)
  {
    unsigned long now = millis();
    double timechange = (double)(now - lastTime);


    currentpos = analogRead(fdbk);
    pos_er = desired - currentpos;          //self explanatory
    errSUM += (pos_er * timechange);
    double dErr = (pos_er - LastERR) / timechange;

//    Serial.print("CURRENT POSITION = ");      //Printing the needed
variables for observation
//    Serial.println(currentpos);
//    Serial.print("DESIRED POSITION = ");
```

```
//    Serial.println(desired);
//    Serial.print("ERROR TO BE COVERED = ");
    Serial.print(pos_er);
  Serial.print("\t");
  Serial.println("127");

    Outputval = kp * pos_er + ki*errSUM + kd*dErr;


    LastERR = pos_er;
    lastTime = now - sampletime;

    //delay(500);


    if (currentpos < right_comp | currentpos <right)
    {
      digitalWrite(InA, LOW);
      analogWrite(InB, 254);
    }
    else if (currentpos > left_comp | currentpos >left)
    {
      digitalWrite(InA, LOW);
      analogWrite(InB, 1);
    }


    if ((currentpos >= right_comp) && (currentpos <= left_comp))
    {
      if ((desired >= right_comp) && (desired <= left_comp))
      {
        if (Outputval > 1)
        {
          digitalWrite(Enable, HIGH);
          digitalWrite(InA, LOW);
          int val_pve = map(Outputval, a, b, 128, 254);
          if (val_pve > 254)
          {
          analogWrite(InB, 254);
//          Serial.print("value sent to uC = ");
//          Serial.println(val_pve);
            Serial.print(pos_er);
            Serial.print("\t");

            Serial.println("254");
          }
          else
          {
            analogWrite(InB, val_pve);
//          Serial.print("value sent to uC = ");
//          Serial.println(val_pve);
            Serial.print(pos_er);
            Serial.print("\t");

            Serial.println(val_pve);
```

```
            }
        }

        if (Outputval < -1)
        {
          digitalWrite(Enable, HIGH);

          digitalWrite(InA, LOW);

          int val_nve = map(Outputval, -a, -b, 126, 1);
          if (val_nve < 1)
          {
          analogWrite(InB, 1);
//          Serial.print("value sent to uC = ");
//          Serial.println(val_nve);
            Serial.print(pos_er);
            Serial.print("\t");

            Serial.println("1");
          }
          else
          {
            analogWrite(InB, val_nve);
//          Serial.print("value sent to uC = ");
//          Serial.println(val_nve);
          Serial.print(pos_er);
          Serial.print("\t");
          Serial.println(val_nve);



            }

            }

      }
    }
  }
  if (pos_er == 0)
  {
    digitalWrite(Enable, HIGH);

    digitalWrite(InA, HIGH);
    analogWrite(InB, 127);
Serial.print("0");
Serial.print("\t");
Serial.println("127");


  }




}
```

## Source Code: Speed Control

```
#include <SPI.h>

byte address = 0x00;
int CS= 10;

void setup()
{
pinMode (CS, OUTPUT);
SPI.begin();
}

void loop()
{

digitalPotWrite(16); //speed value
delay(10);

}

int digitalPotWrite(int value)
{
digitalWrite(CS, LOW);
SPI.transfer(address);
SPI.transfer(value);
digitalWrite(CS, HIGH);
}
```