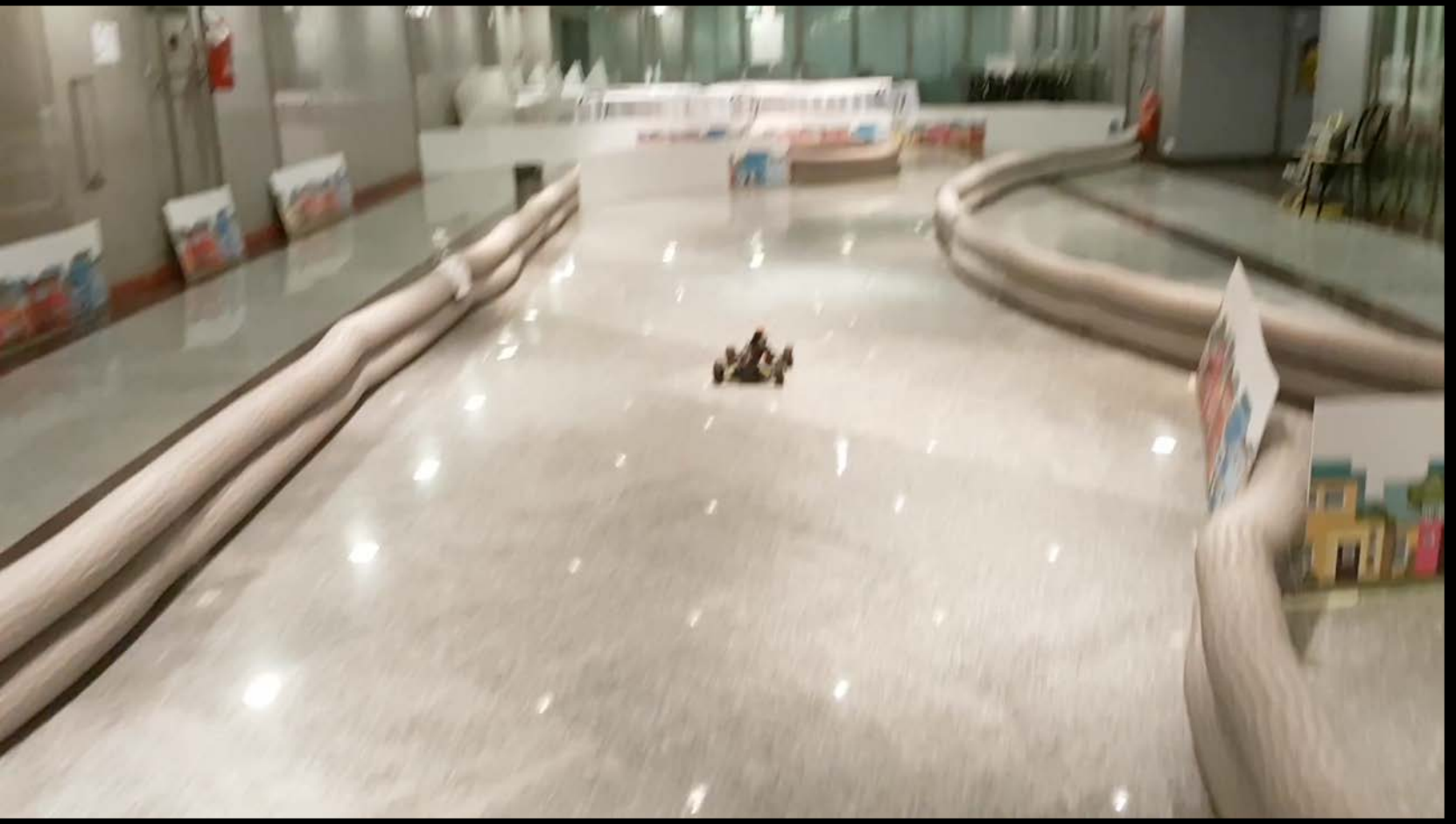
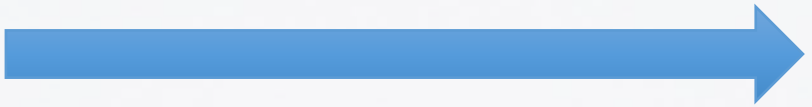


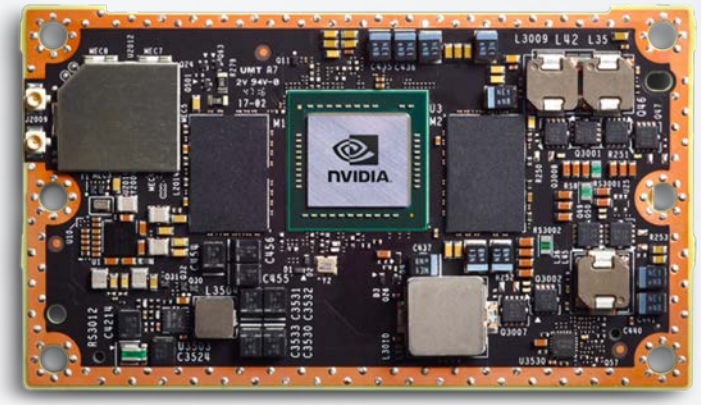
F1/10 Autonomous Racing

Reactive Methods
Part 1
Wall Following

Madhur Behl
(University of Virginia)

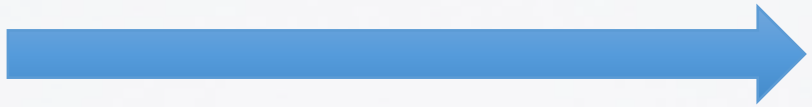
Previously

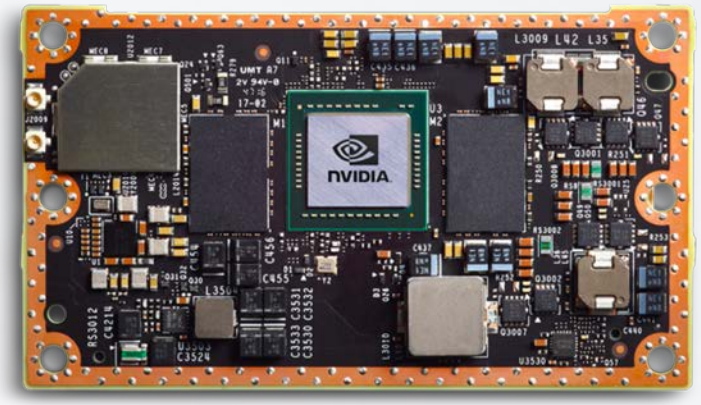




ssh nvidia@192.168.1.1

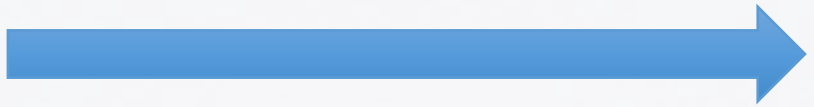
TeleOp Test

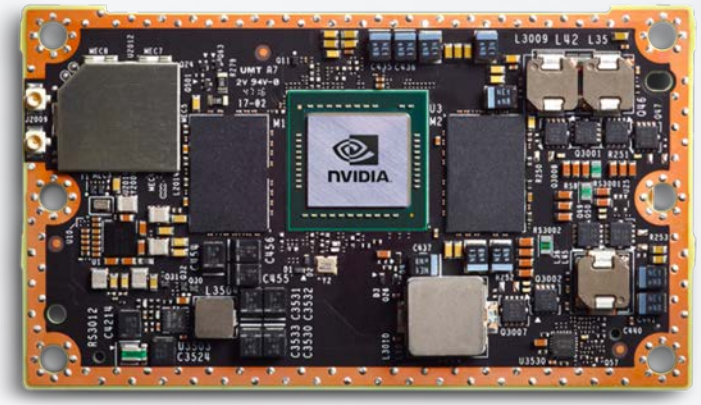




roslaunch move_base move_base.launch

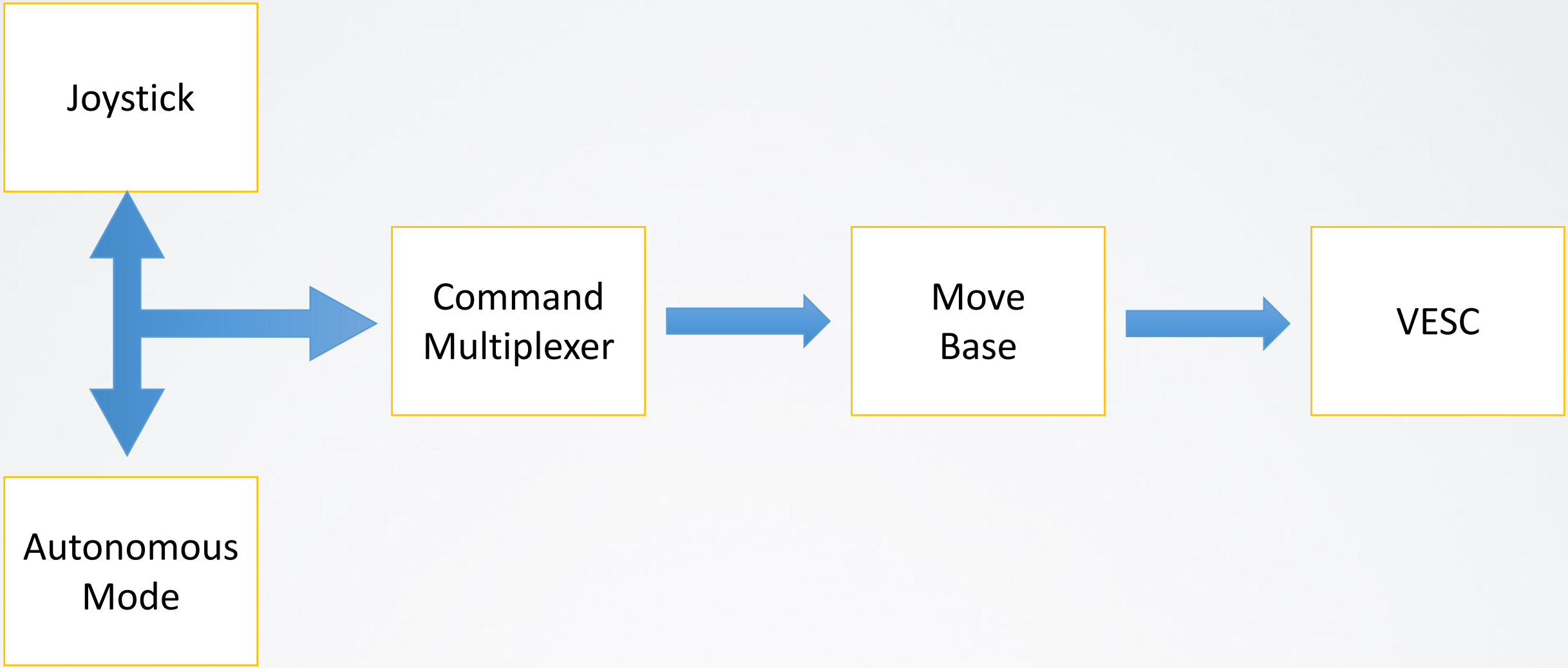
Lets see how it works…



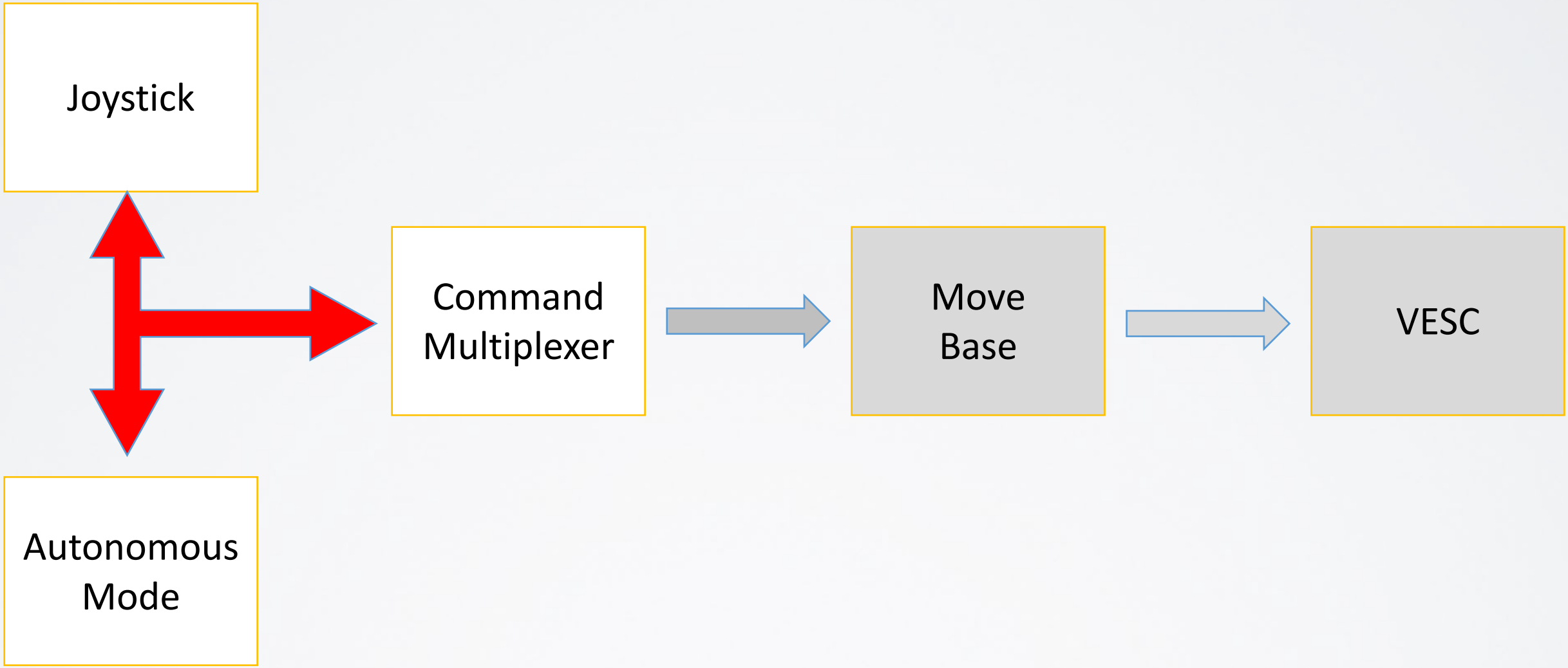


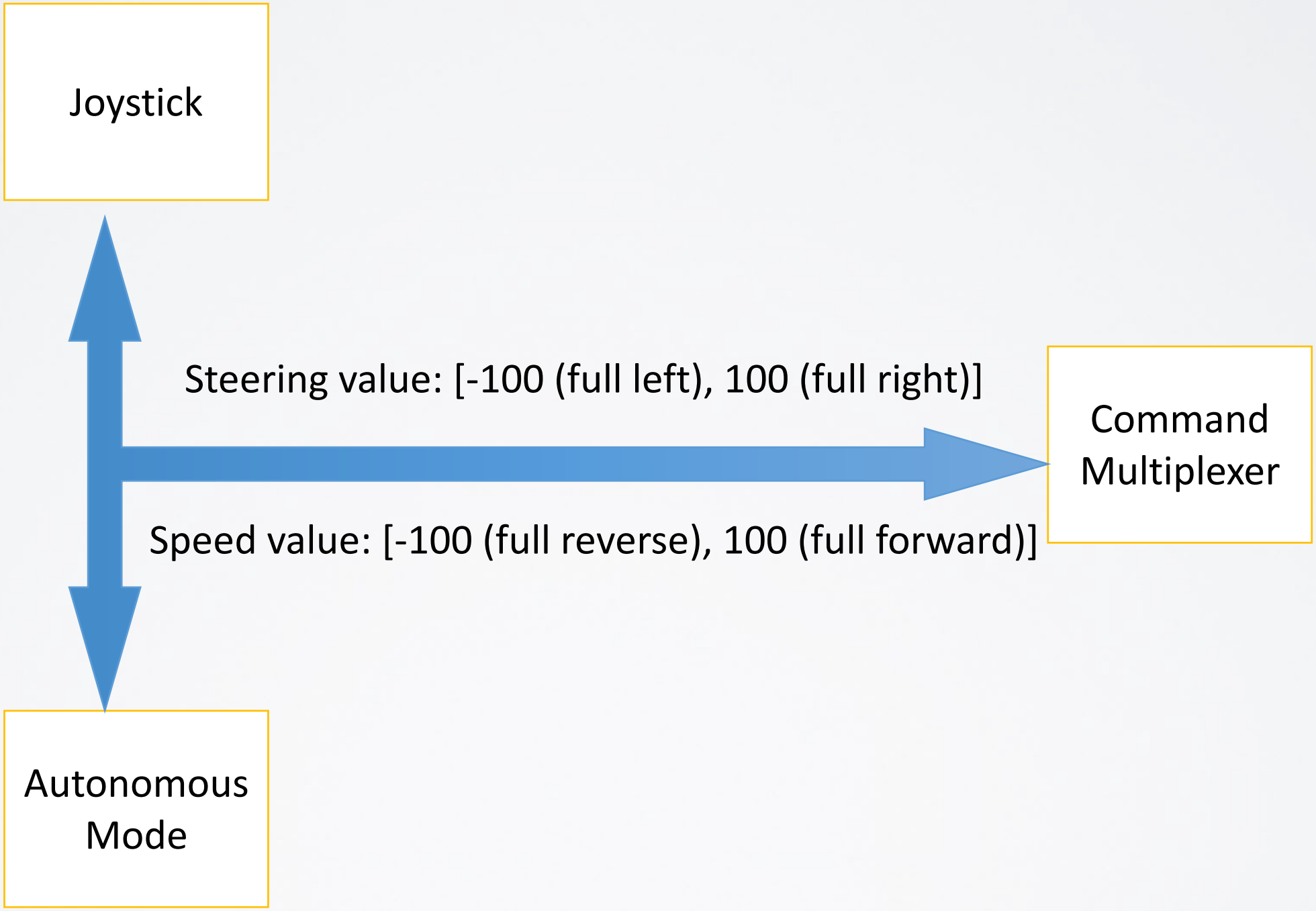
roslaunch move_base move_base.launch

On the Jetson TX2



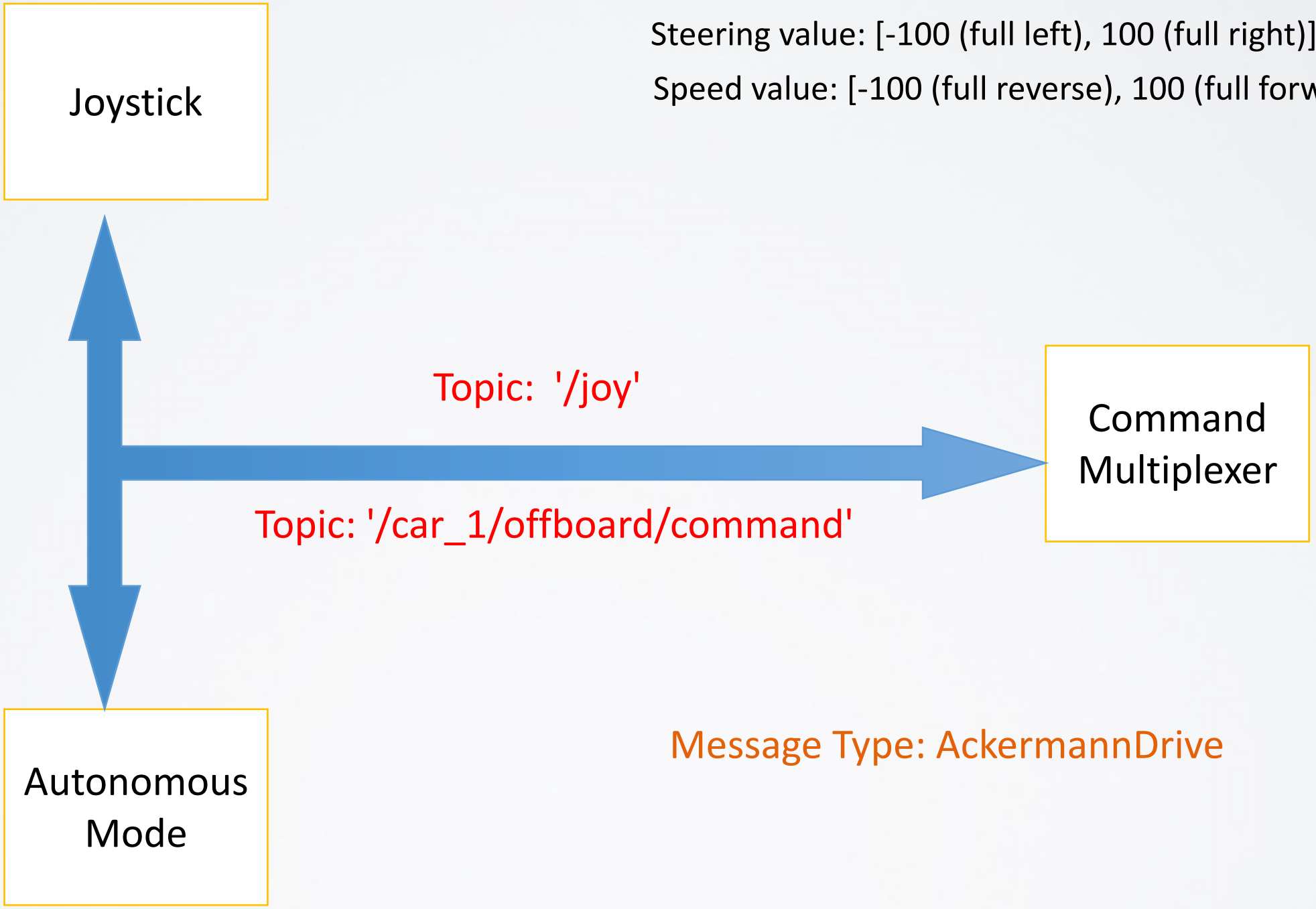
On the Jetson TX2




Joystick
Steering value: [-100 (full left), 100 (full right)]
Command Multiplexer
Speed value: [-100 (full reverse), 100 (full forward)]
Autonomous Mode

Steering value: [-100 (full left), 100 (full right)]

Speed value: [-100 (full reverse), 100 (full forward)]



Message Type: AckermannDrive

ackermann_msgs/AckermannDrive Message

File: `ackermann_msgs/AckermannDrive.msg`

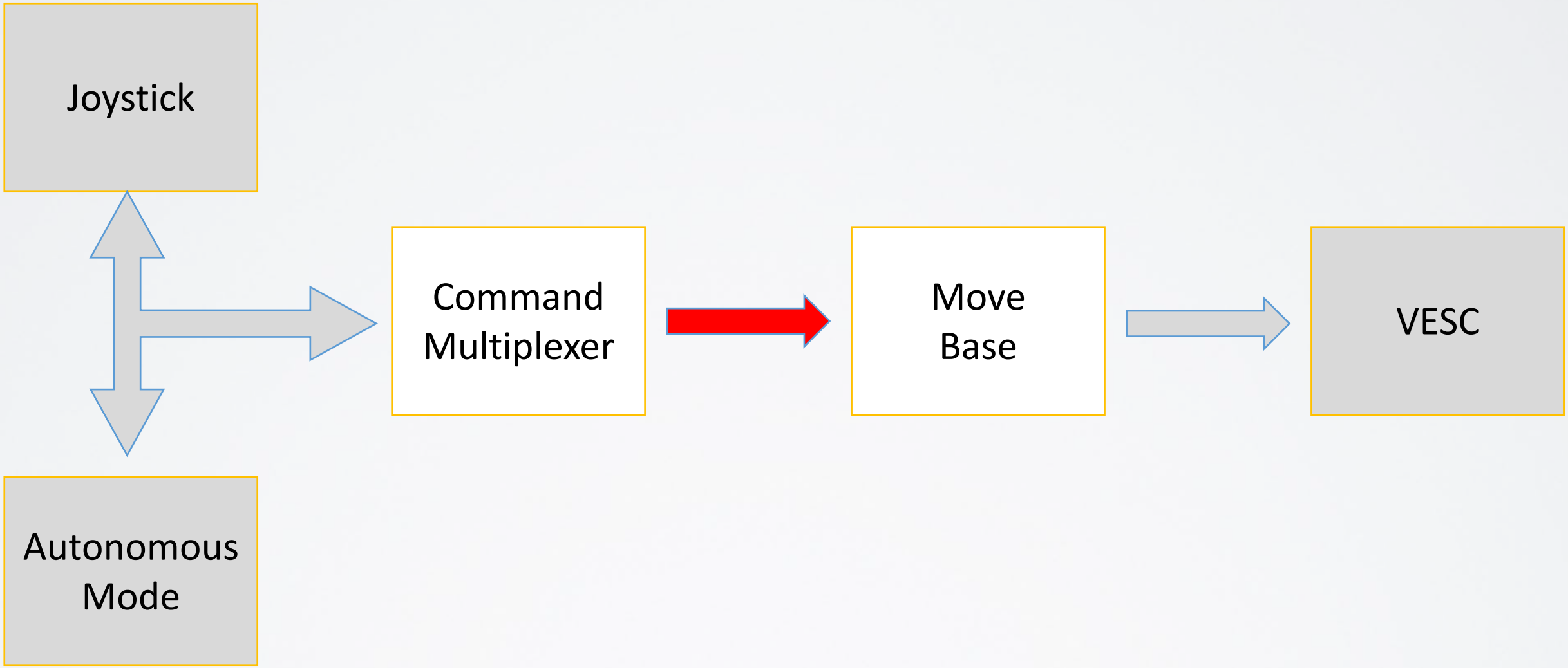
Raw Message Definition

```
float32 steering_angle          [-100,100]
float32 steering_angle_velocity [-100,100]
float32 speed
float32 acceleration
float32 jerk
```

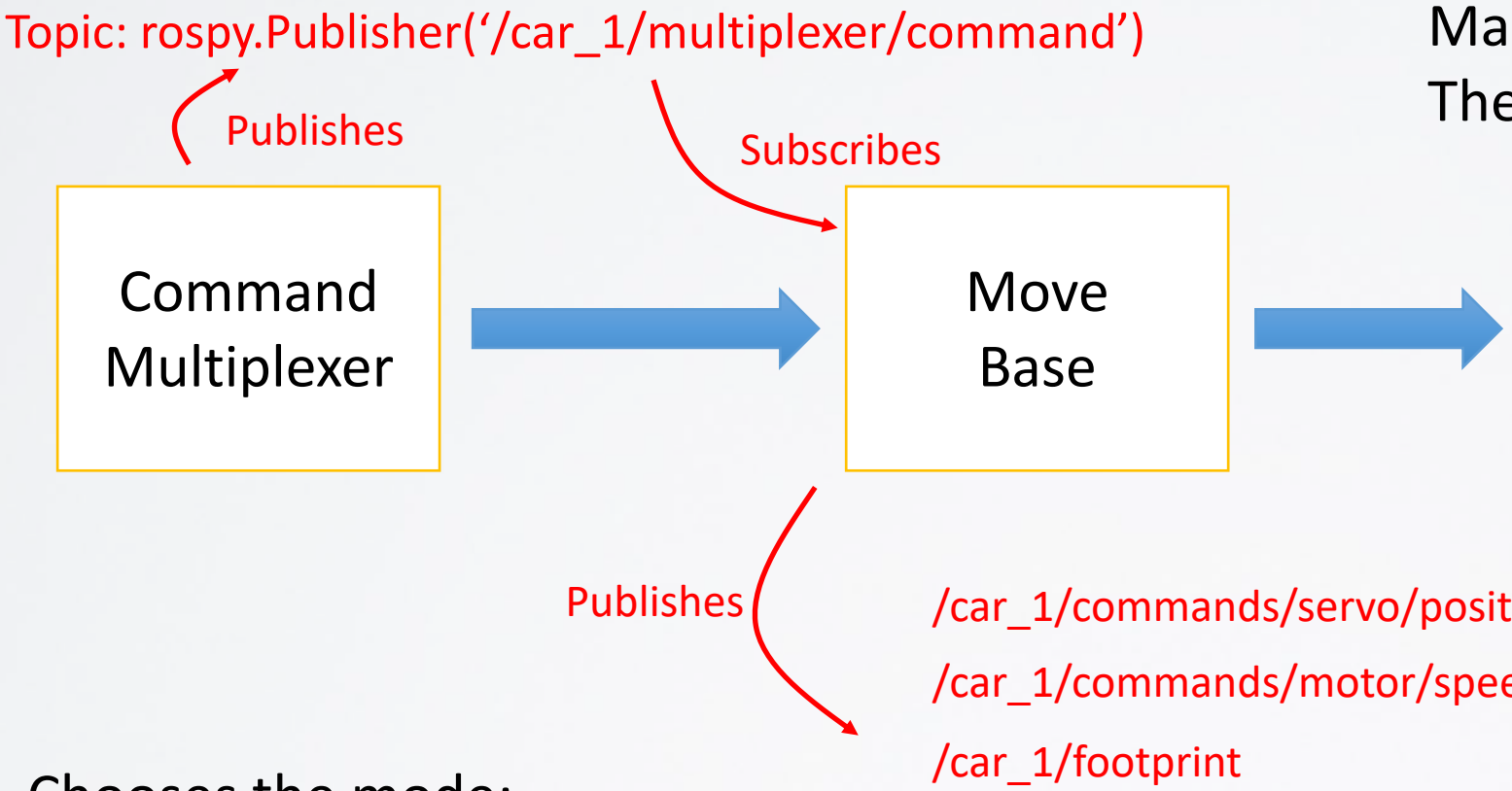
Only two fields are used

These do not matter and are set to 0 by default.

On the Jetson TX2



Steering and Speed Values
[-100,100] for both



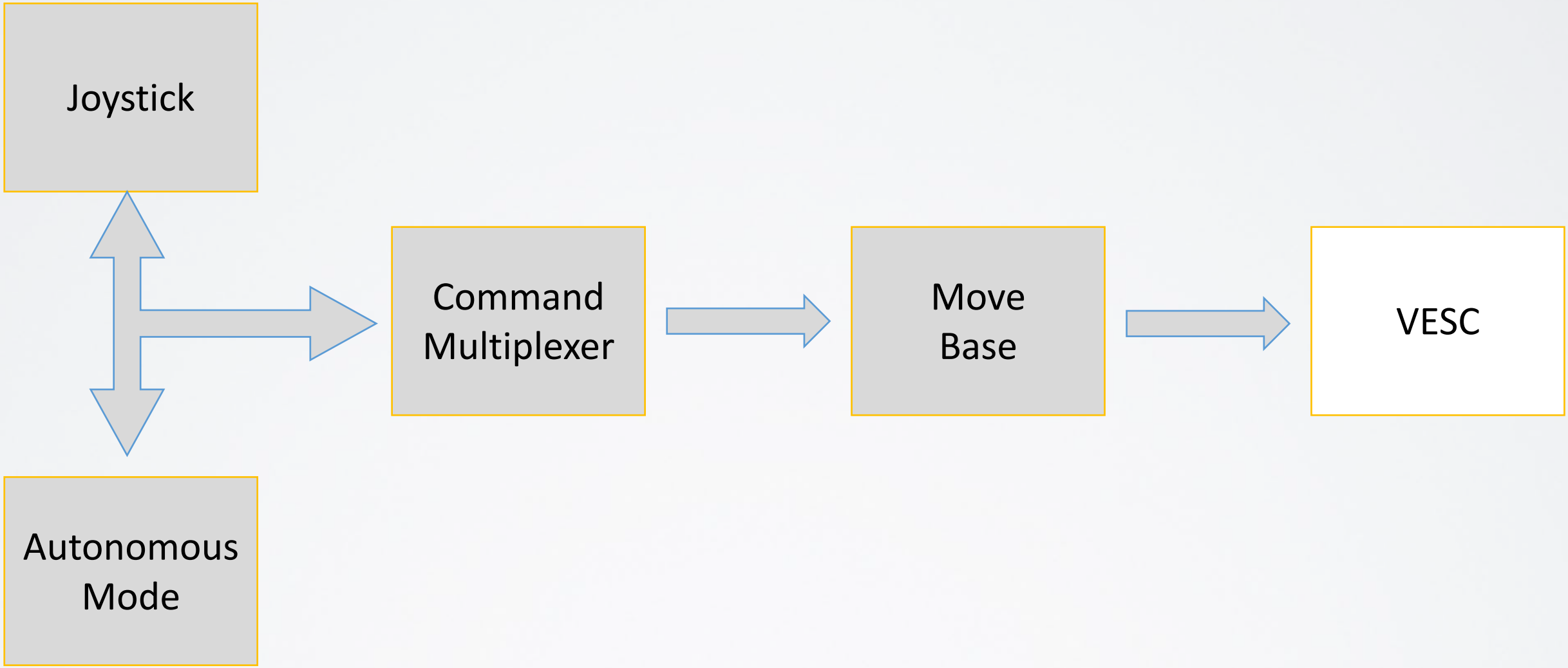
Maps the range into float64 values for The VESC driver:

Steering:
- [-100,100] is mapped to [0,1]

Speed:
- [-100,100] is mapped to [-20000,20000] RPM

Chooses the mode:
Autonomous
Or
TeleOp

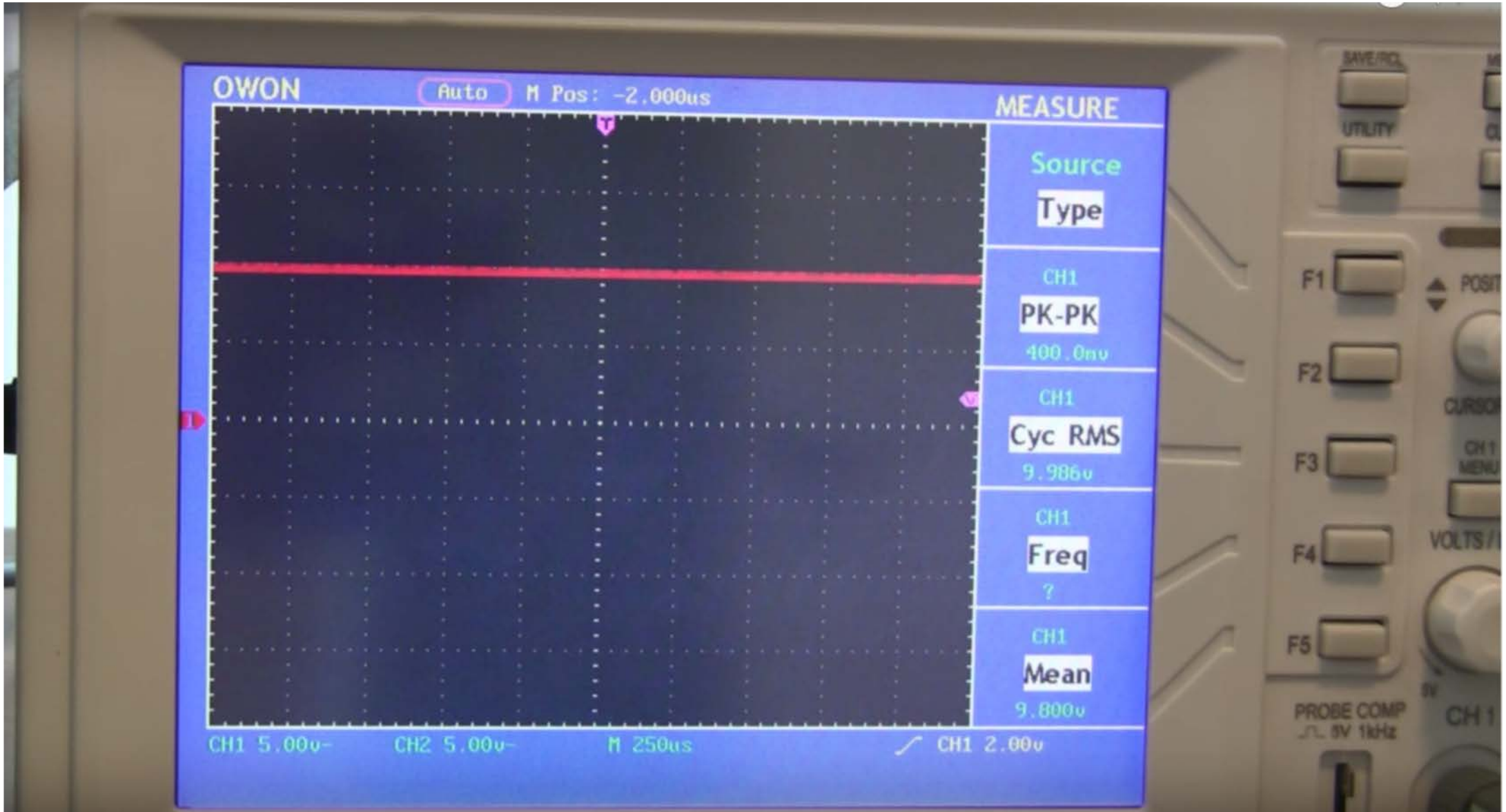
On the Jetson TX2



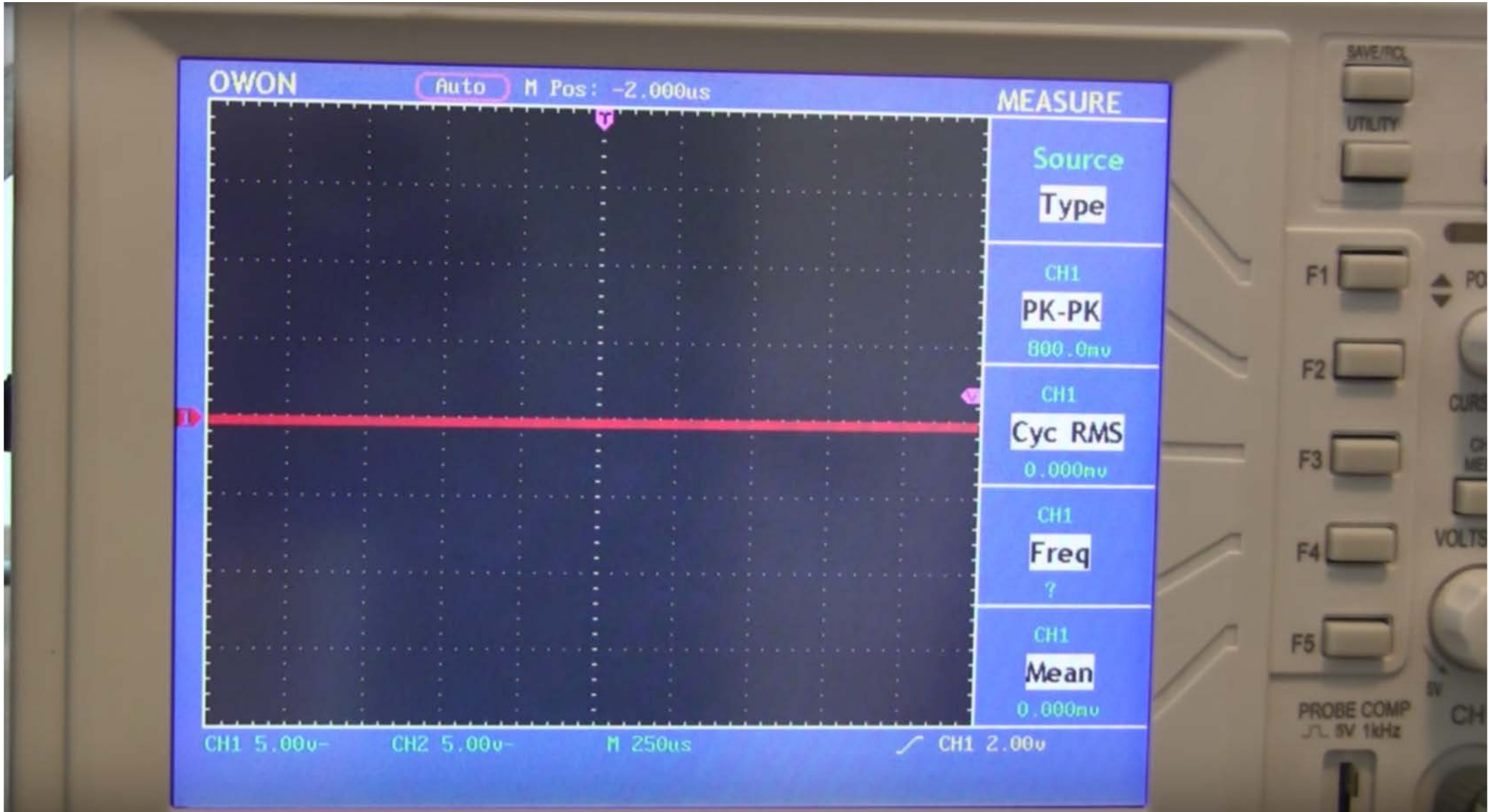
What is PWM – Pulse Width Modulation

- Output signal alternates between on and off within specified period
- Controls power received by a device
- The voltage seen by the load is directly proportional to the source voltage

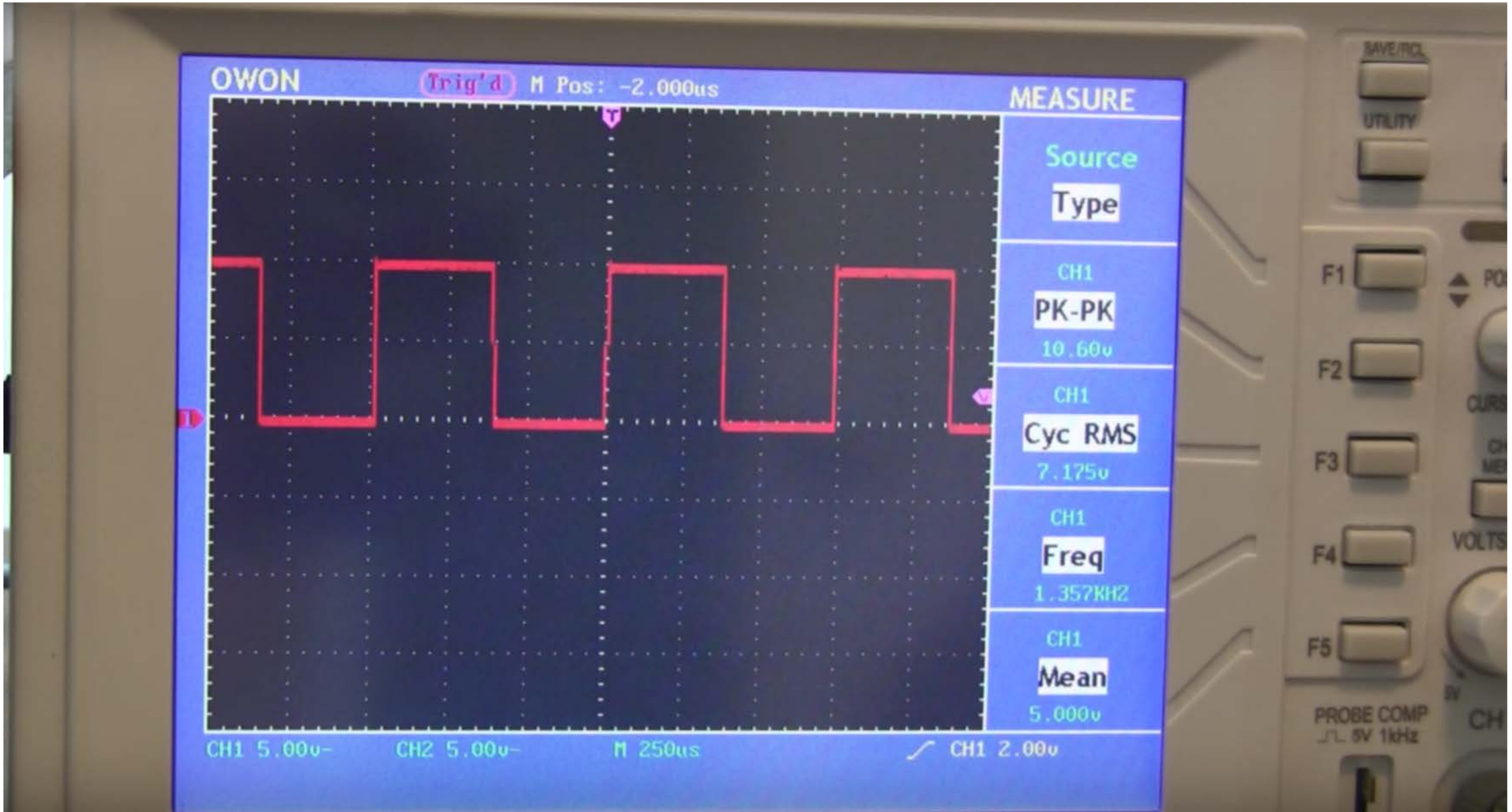
Here is what 10V DC looks like..



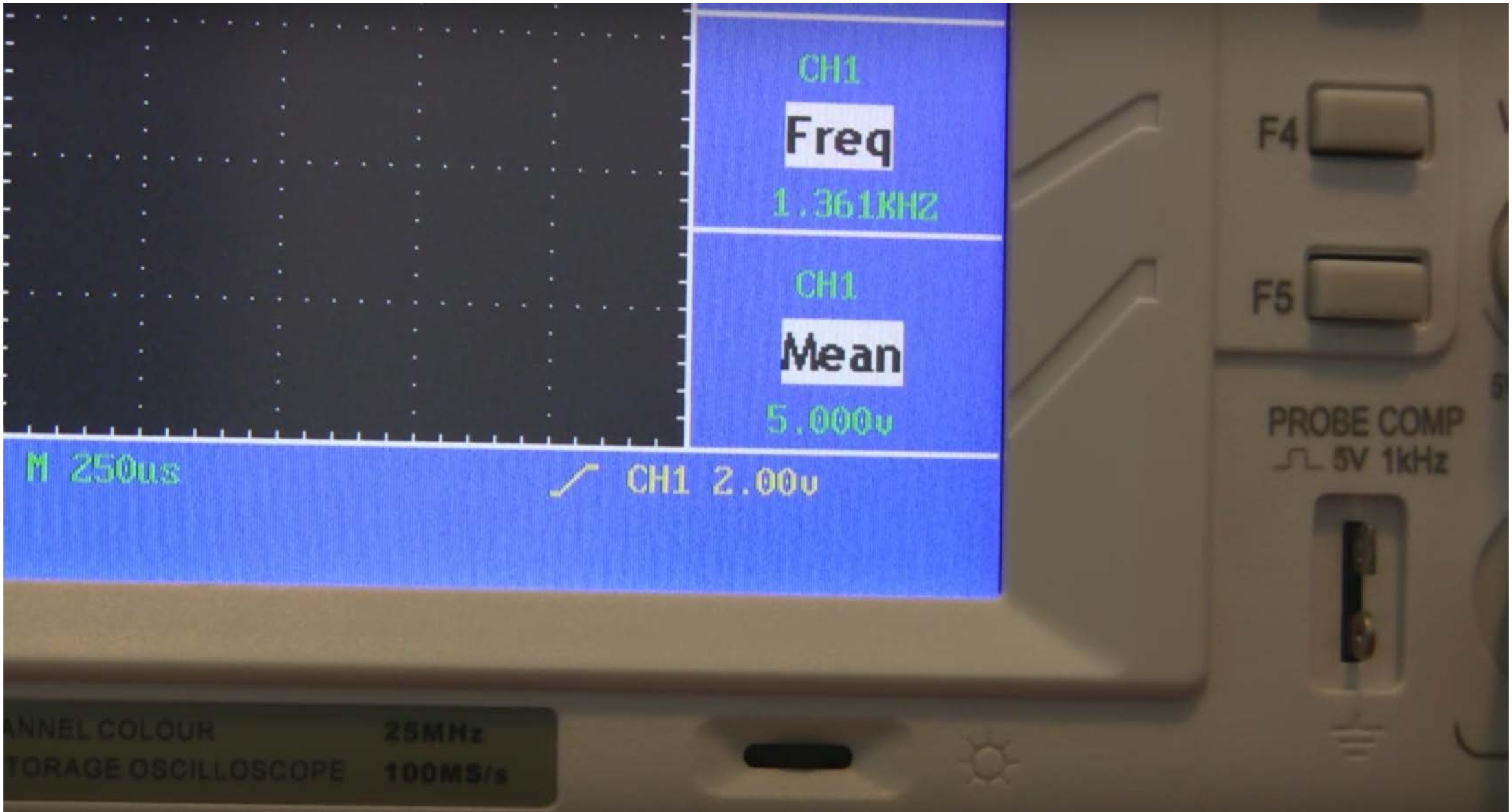
Here is what 0V DC looks like..



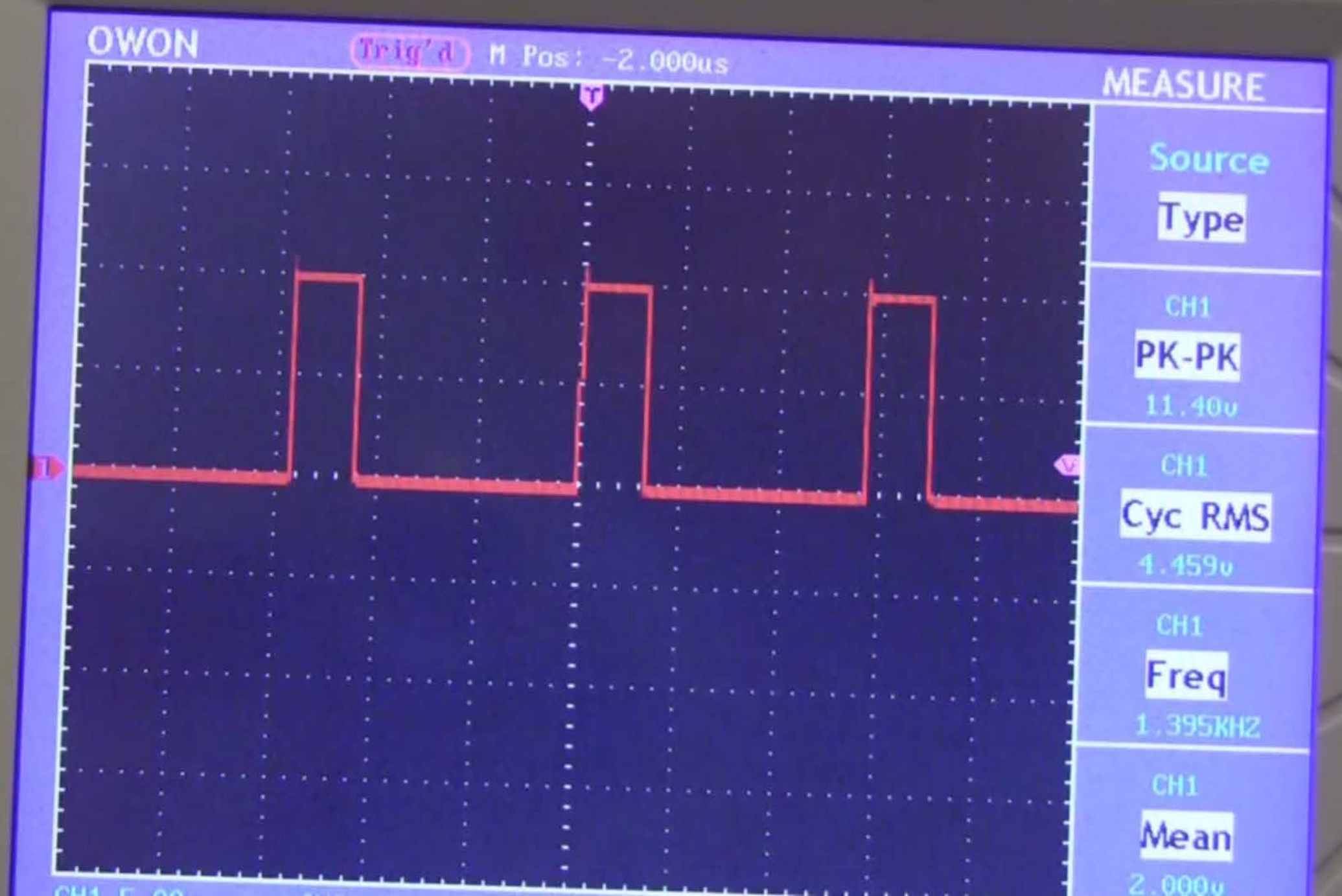
Switch 10V ON half the time : 50% Duty Cycle

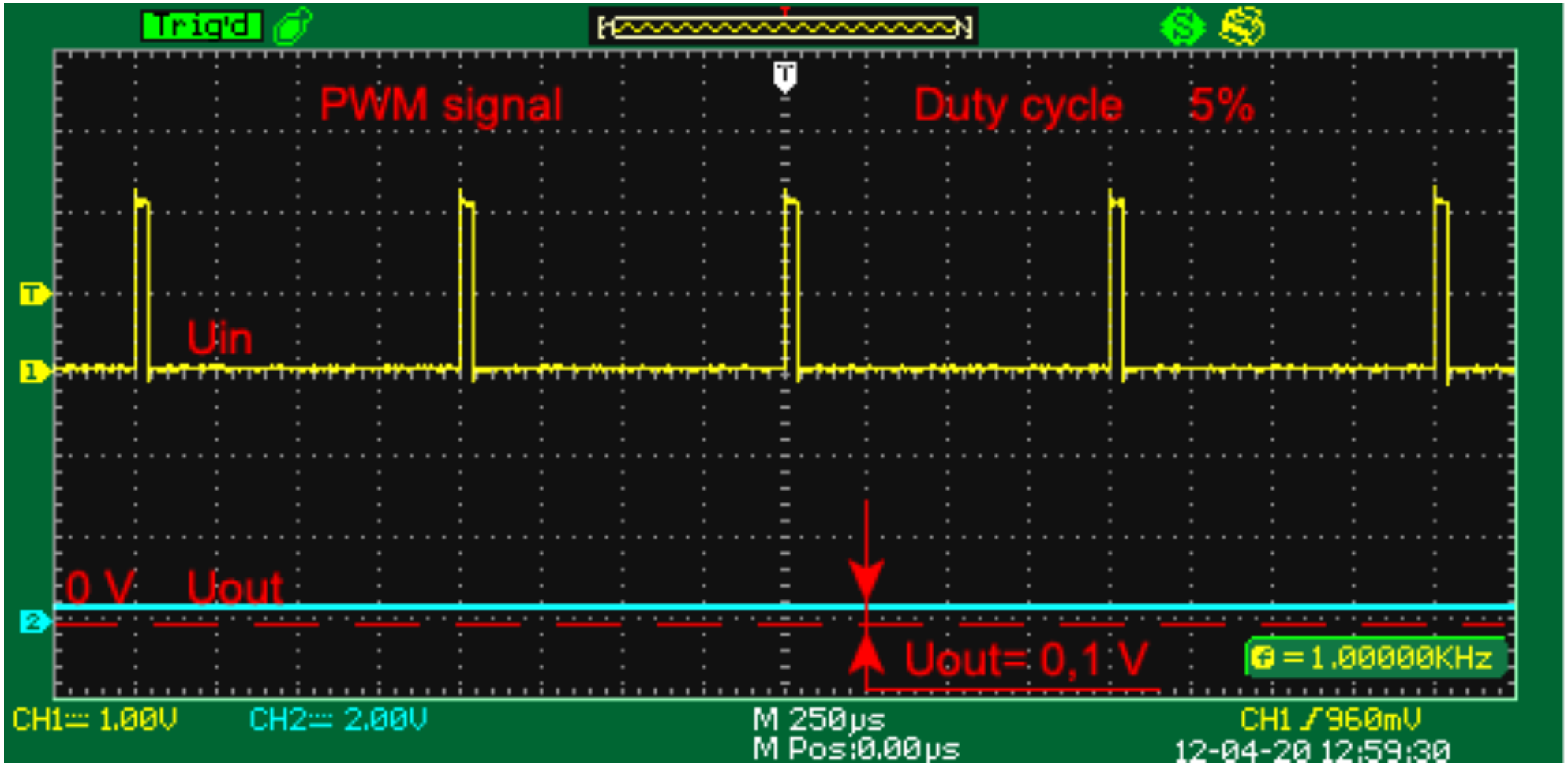


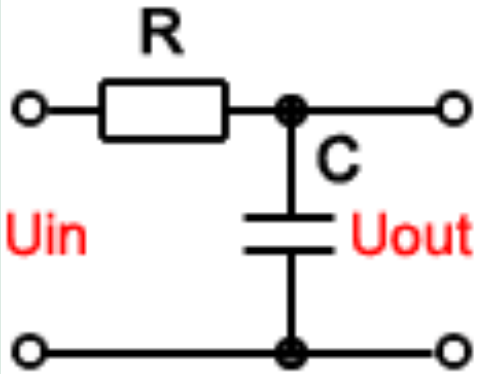
10V DC: 50% duty cycle → average of 5V DC



Here is a case with 10% duty cycle..

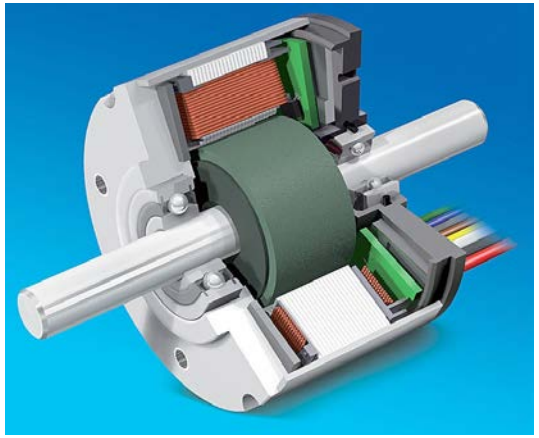

OWON
Trig'd
M Pos: -2.000us
MEASURE
Source
Type
CH1
PK-PK
11.40v
CH1
Cyc RMS
4.459v
CH1
Freq
1.395KHz
CH1
Mean
2.000v


Trig'd
PWM signal
Duty cycle 5%
Uin
0 V Uout
Uout= 0,1 V
f = 1,00000KHz
CH1 1.00V
CH2 2.00V
M 250µs
M Pos:0.00µs
CH1 960mV
12-04-20 12:59:30


R
C
Uin
Uout

Application to DC motors

- The voltage supplied to a DC motor is proportional to the duty cycle
- Both brushed and brushless motors can be used with PWM
- Both analog and digital control techniques and components are available



VESC

move_base / config / vesc.config

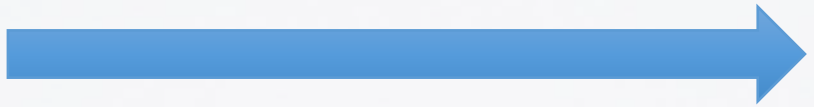
28 lines (25 sloc) | 587 Bytes

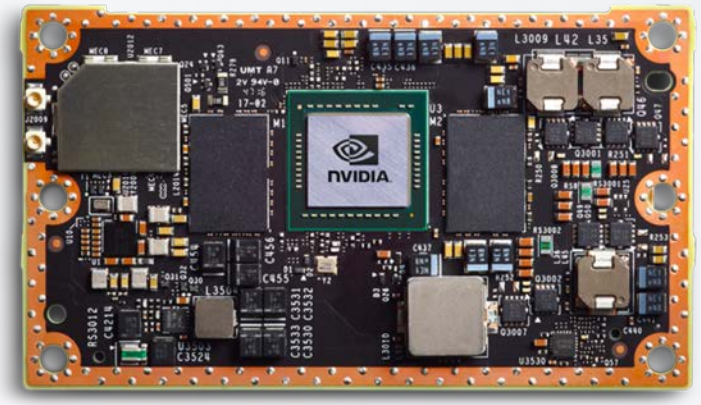
```
speed_to_erpm_gain: 7500
speed_to_erpm_offset: 0.0

tachometer_ticks_to_meters_gain: 0.00225
max_servo_speed: 3.2
servo_smoother_rate: 75.0
max_acceleration: 2.5
throttle_smoother_rate: 75.0
steering_angle_to_servo_gain: -1.15
steering_angle_to_servo_offset: 0.5

vesc_to_odom/publish_tf: False
wheelbase: .325

vesc_driver:
  port: /dev/ttyACM1
  duty_cycle_min: 0.0
  duty_cycle_max: 0.0
  current_min: 0.0
  current_max: 20.0
  brake_min: -20000.0
  brake_max: 20000.0
  speed_min: -20000
  speed_max: 20000
  position_min: 0.0
  position_max: 0.0
  servo_min: 0.05
  servo_max: 0.95
```

vesc_driver.py

Sends PWM commands to Drive Motor and Steering Servo

TeleOp Test





roslaunch move_base move_base.launch

Let us study **move_base.launch**

```
5     <!-- agent properties -->
6     <arg          name          = 'car_name'
7                   default       = 'car_1'/>
8     <arg          name          = 'listen_offboard'
9                   default       = 'false'/>
```

listen_offboard : false -> TeleOp mode, true -> Autonomous Mode

Let us study move_base.launch

```
58      <!-- move_base node -->
59      <node           name            = 'move_base'
60                      pkg             = 'move_base'
61                      type            = 'move_base.py'
62                      args            = '$(arg car_name)'/> </group>
63
```

move_base.py node is run

Let us study move_base.launch

```
51    <!-- joystick based command_multiplexer node -->
52    <node          name              = 'command_multiplexer'
53                   pkg               = 'move_base'
54                   type              = 'command_multiplexer.py'
55                   args              = '$(arg car_name) $(arg listen_offboard)'
56                   output            = 'screen'/>
```

Takes the mode as input argument

Command_multiplex.py node is run

Let us study move_base.launch

```
41    <!-- static transform between base_link and laser -->
42    <node          name                = 'base_laser_link'
43                   pkg                 = 'tf'
44                   type                = 'static_transform_publisher'
45                   args                = '0.15 0.0 0.0
46                                          0.0 0.0 0.0
47                                          $(arg car_name)_base_link
48                                          $(arg car_name)_laser
49                                          10'/>
```

Static transform between base_link
And laser

Let us study move_base.launch

```
30      <!-- URG LiDAR node -->
31      <node           name            = 'urg_laser_scanner'
32                      pkg             = 'urg_node'
33                      type            = 'urg_node'>
34      <param          name            = 'serial_port'
35                      value           = '/dev/ttyACM0'/>
36      <param          name            = 'serial_baud'
37                      value           = '115200'/>
38      <param          name            = 'frame_id'
39                      value           = '$(arg car_name)_laser'/> </node>
```

Note the port it looks for

Let us study move_base.launch

```
18    <!-- launch VESC node driver -->
19    <node          name              = 'vesc_driver'
20                   pkg               = 'vesc_driver'
21                   type              = 'vesc_driver_node'>
22    <param         name              = 'port'
23                   value             = '/dev/ttyACM1'/> </node>
```

Note the port it looks for

Let us study move_base.launch

```
11      <group          ns                  = '$(arg car_name)'>
12      <!-- load VESC odometry parameters -->
13      <arg            name                = 'vesc_config'
14                      default             = '$(find move_base)/config/vesc.config'/>
15      <rosparam       file                = '$(arg vesc_config)'
16                      command             = 'load'/>
```

Let us study move_base.launch

```
25      <!-- launch VESC odometry calculator -->
26      <node           name                = 'vesc_to_odom'
27                      pkg                 = 'vesc_ackermann'
28                      type                = 'vesc_to_odom_node'/>
```

move_base / src / command_multiplexer.py

```
rospy.init_node('command_multiplexer', anonymous = True)
if listen_offboard == 'true':
    rospy.Subscriber('/{}/offboard/command'.format(car_name), AckermannDrive, offboard_callback)
rospy.Subscriber('/joy', Joy, joy_command_callback)
rospy.spin()
```

move_base / src / command_multiplexer.py

```
multiplexer_pub       = rospy.Publisher('/{}/multiplexer/command'.format(car_name), AckermannDrive, queue_size = 1)
offboard_command      = AckermannDrive()
```

move_base / src / command_multiplexer.py

ackermann_msgs/AckermannDrive Message

File: `ackermann_msgs/AckermannDrive.msg`

Raw Message Definition

```
float32 steering_angle
float32 steering_angle_velocity
float32 speed
float32 acceleration
float32 jerk
```

move_base / src / move_base.py

```
1  #!/usr/bin/env python
2
3  import rospy
4  import sys
5
6  from std_msgs.msg import Float64
7  from ackermann_msgs.msg import AckermannDrive
8  from geometry_msgs.msg import Point32
9  from geometry_msgs.msg import PolygonStamped
```

move_base / src / move_base.py

```
rospy.init_node('move_base', anonymous = True)
rospy.Subscriber('/{}/multiplexer/command'.format(car_name), AckermannDrive, command_callback)
```

move_base / src / move_base.py

```
76    def command_callback(data):
77        angle_req = Float64()
78        speed_req = Float64()
79        angle_req.data = output_angle_mixer(data.steering_angle)
80        speed_req.data = output_speed_mixer(data.speed)
81        angle_pub.publish(angle_req)
82        speed_pub.publish(speed_req)
83        footprint_visualizer()
```

move_base / src / move_base.py

```
32   angle_min_rel      = -100.0          # max left command
33   angle_max_rel      = 100.0           # max right command
34   angle_min_abs      = 0.0             # max left VESC servo
35   angle_max_abs      = 1.0             # max right VESC servo
36
37   speed_min_rel      = -100.0          # min speed command
38   speed_max_rel      = 100.0           # max speed command
39   speed_min_abs      = -20000.0        # min speed VESC motor
40   speed_max_abs      = 20000.0         # max speed VESC motor
41   speed_change_step = 2000.0           # acceleration P-type control
```

move_base / src / move_base.py

```
16  angle_pub         = rospy.Publisher('/{}/commands/servo/position'.format(car_name), Float64, queue_size = 1)
17  speed_pub         = rospy.Publisher('/{}/commands/motor/speed'.format(car_name),    Float64, queue_size = 1)
18  footprint_pub     = rospy.Publisher('/{}/footprint'.format(car_name), PolygonStamped, queue_size = 1)
```

move_base / src / move_base.py

```
52    def footprint_visualizer():
53        global seq
54        footprint.header.seq = seq
55        seq = seq + 1
56        footprint.header.stamp = rospy.Time.now()
57        footprint_pub.publish(footprint)
```

Next:

Wall following
PID control

Perception – Wall following

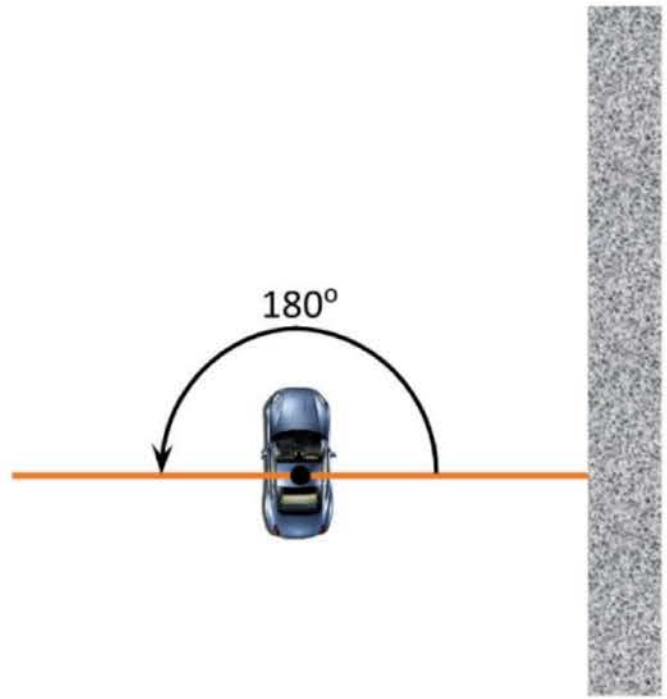
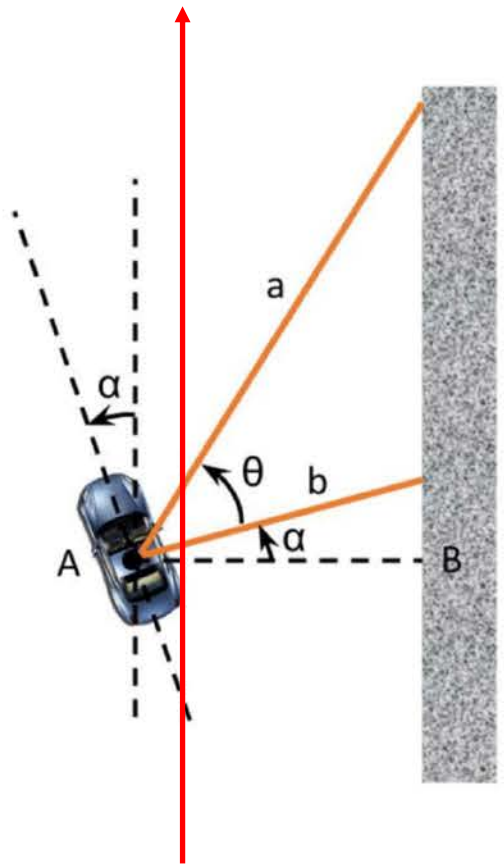


Figure 1: Lidar scan angles

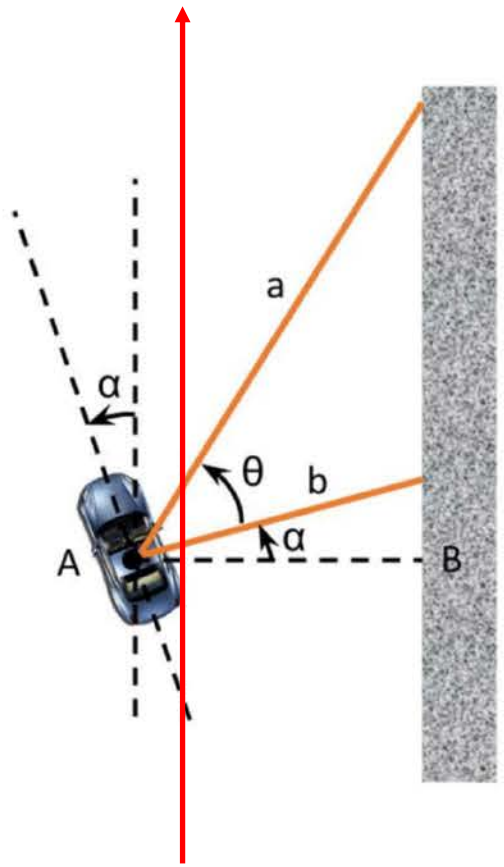
Pick two LIDAR rays facing right – One at 0° and one at θ°



$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta) - b}{a\sin(\theta)}\right)$$

$$AB = b\cos(\alpha)$$

Pick two LIDAR rays facing right – One at 0° and one at θ°

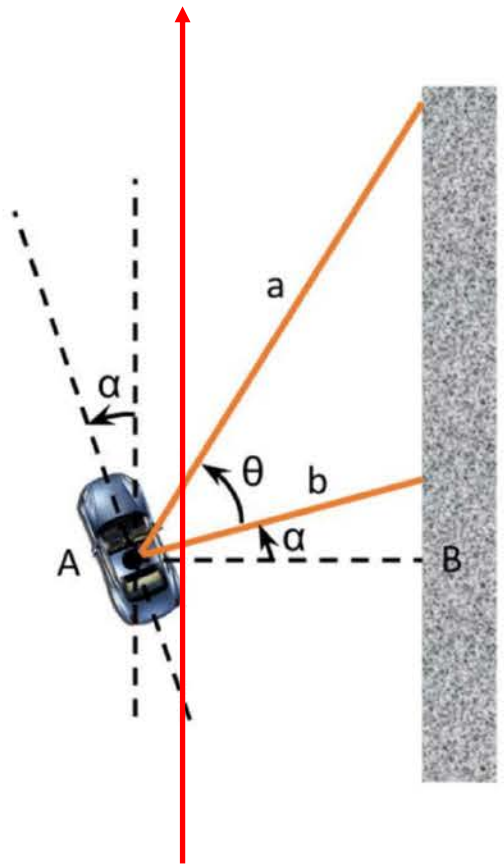


$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta) - b}{a\sin(\theta)}\right)$$

$$AB = b\cos(\alpha)$$

Error = desired trajectory – AB ?

Pick two LIDAR rays facing right – One at 0° and one at θ°



$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta) - b}{a\sin(\theta)}\right)$$

$$AB = b\cos(\alpha)$$

Error = desired trajectory – AB ?

Not quite

Account for the forward motion of the car

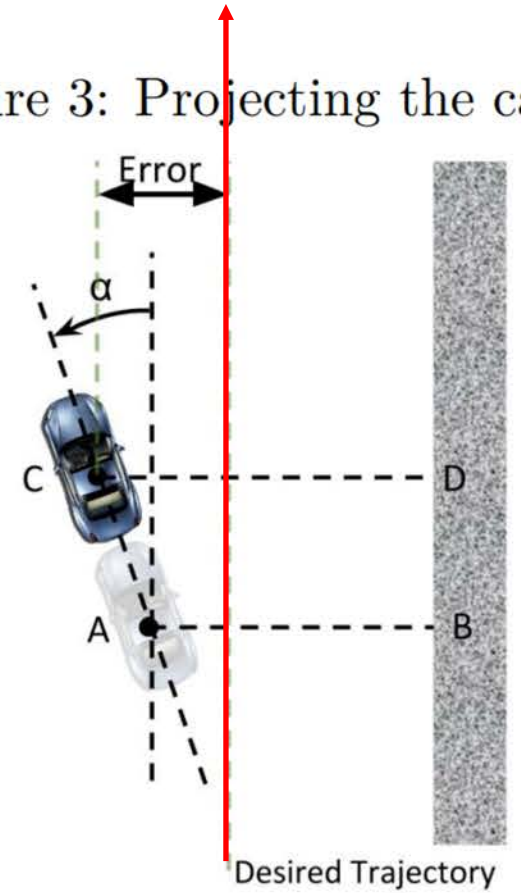


Figure 3: Projecting the car future in time

$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta)-b}{a\sin(\theta)}\right)$$

$$AB = b\cos(\alpha)$$

$$CD = AB + AC\sin(\alpha)$$

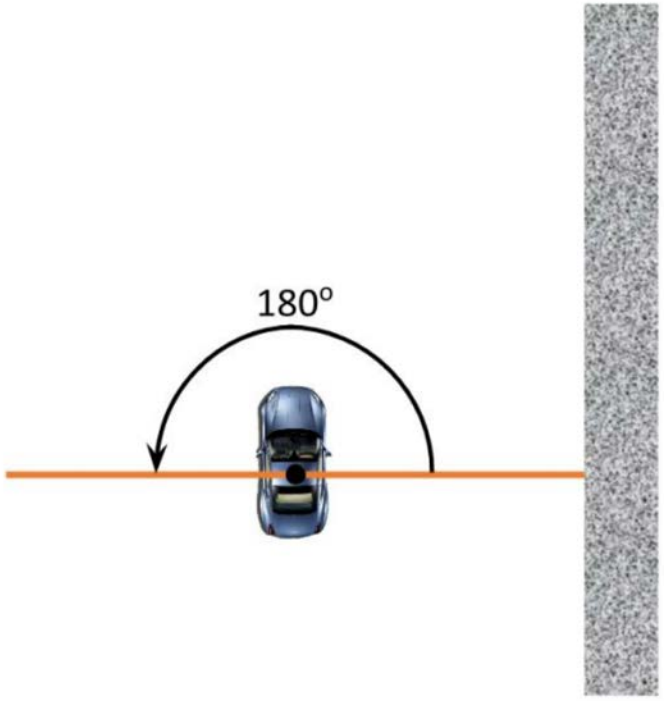
Error = desired trajectory – CD

PID Steering Control

$$V_\theta = K_p \times e(t) + K_d \frac{de(t)}{dt}$$

$$V_\theta = K_p \times error + K_d \times previous\ error - current\ error$$

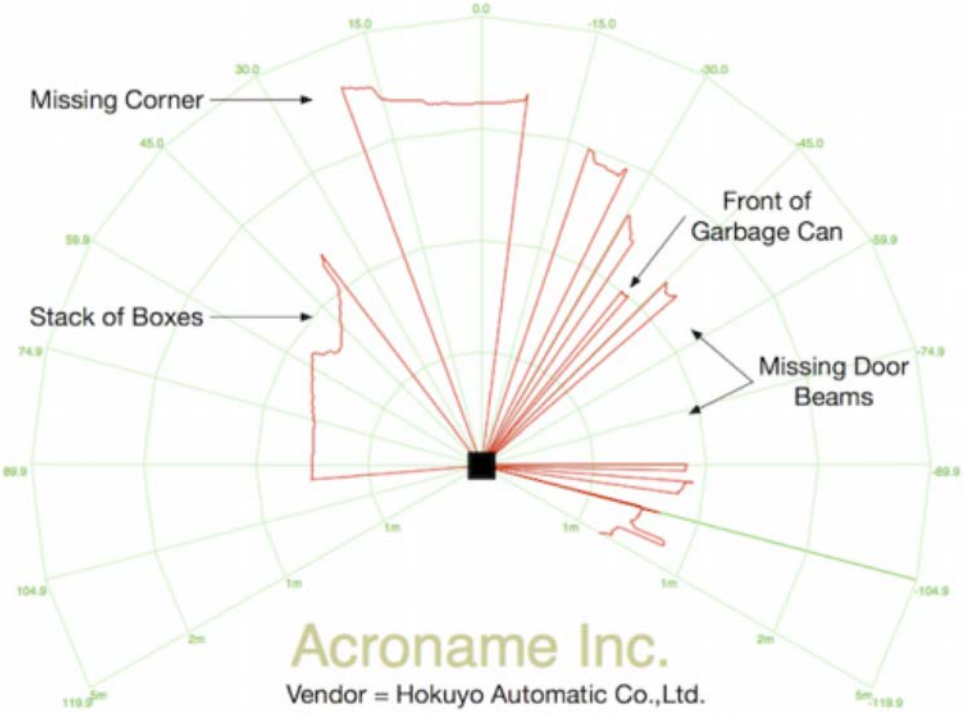
$$steering\ angle = steering\ angle - V_\theta$$



Field of View

← Assumption

Reality →



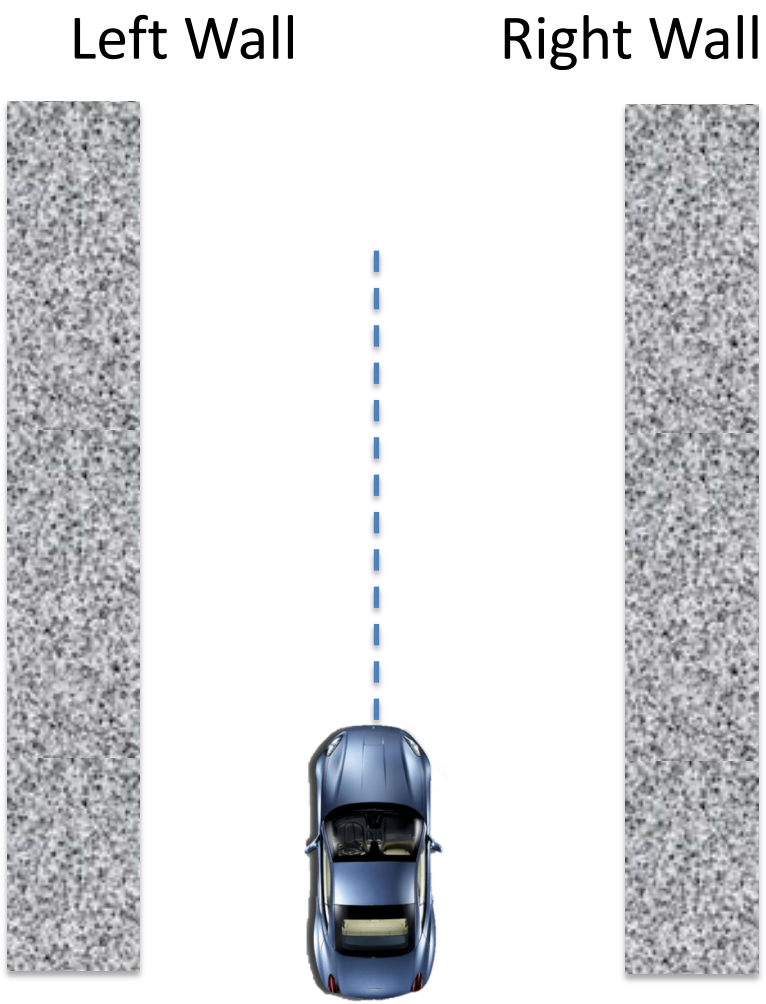
Acroname Inc.

Vendor = Hokuyo Automatic Co.,Ltd.
Model = URG-04LX-UG01 (Simple-URG)
Firmware = 3.4.03
Protocol = SCIP 2.0
Serial number = H1311977

Control

Proportional, **I**ntegral, **D**erivative control

PID control: objectives

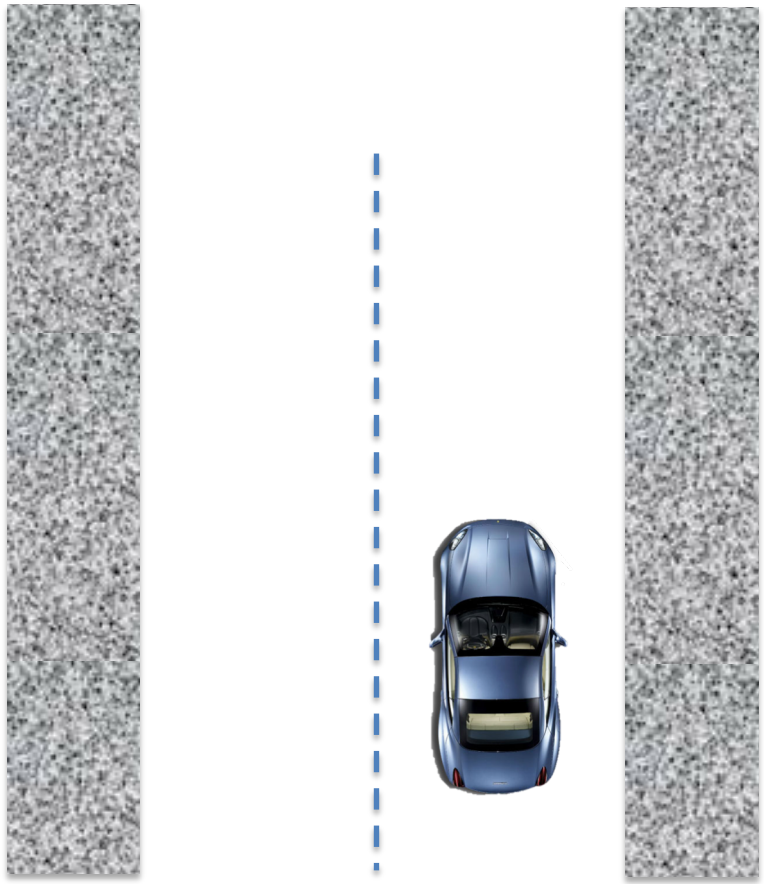


Control objective:

1) keep the car driving along the centerline,

2) parallel to the walls.

PID control: objectives

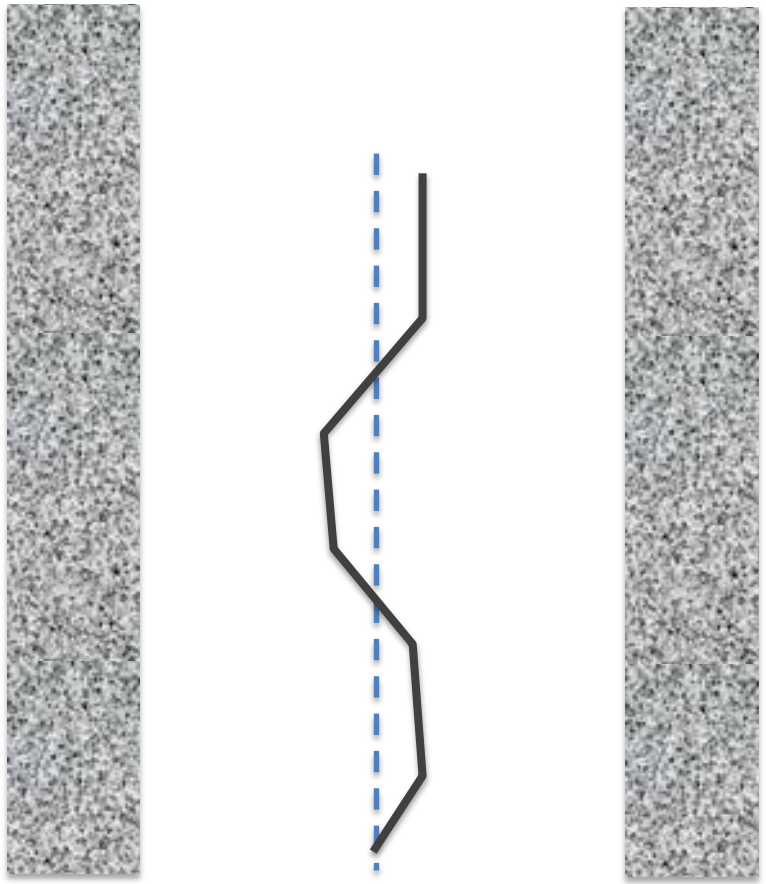


Control objective:

~~1) keep the car driving along the centerline,~~

2) parallel to the walls.

PID control: objectives

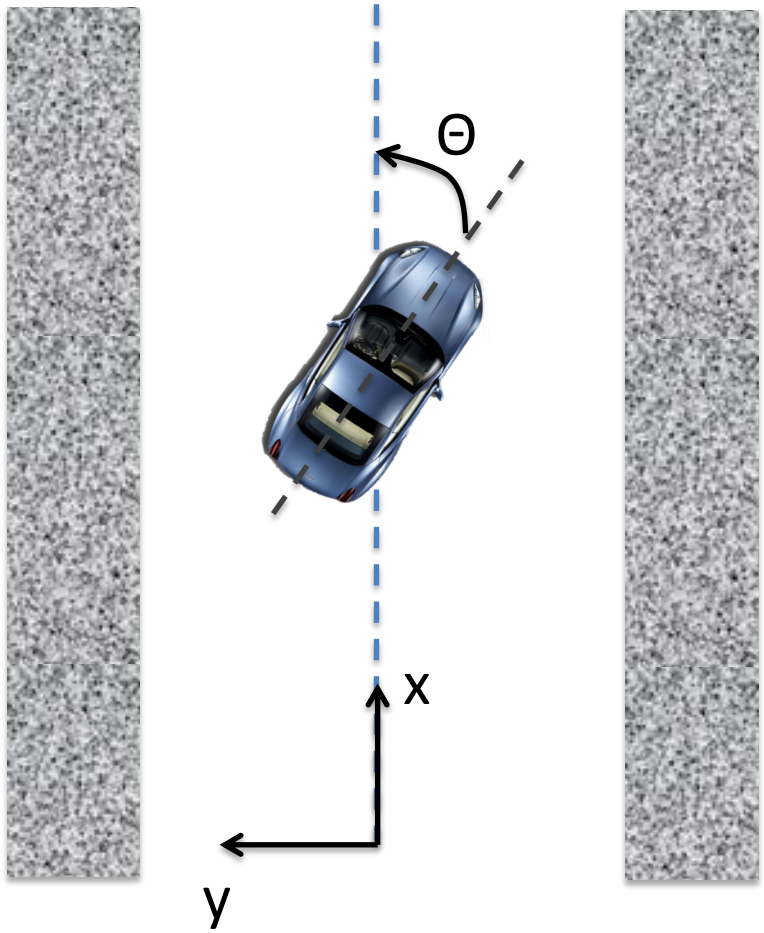


Control objective:

1) keep the car driving (roughly) along the centerline,

~~2) parallel to the walls.~~

PID control: control objectives



Control objective:

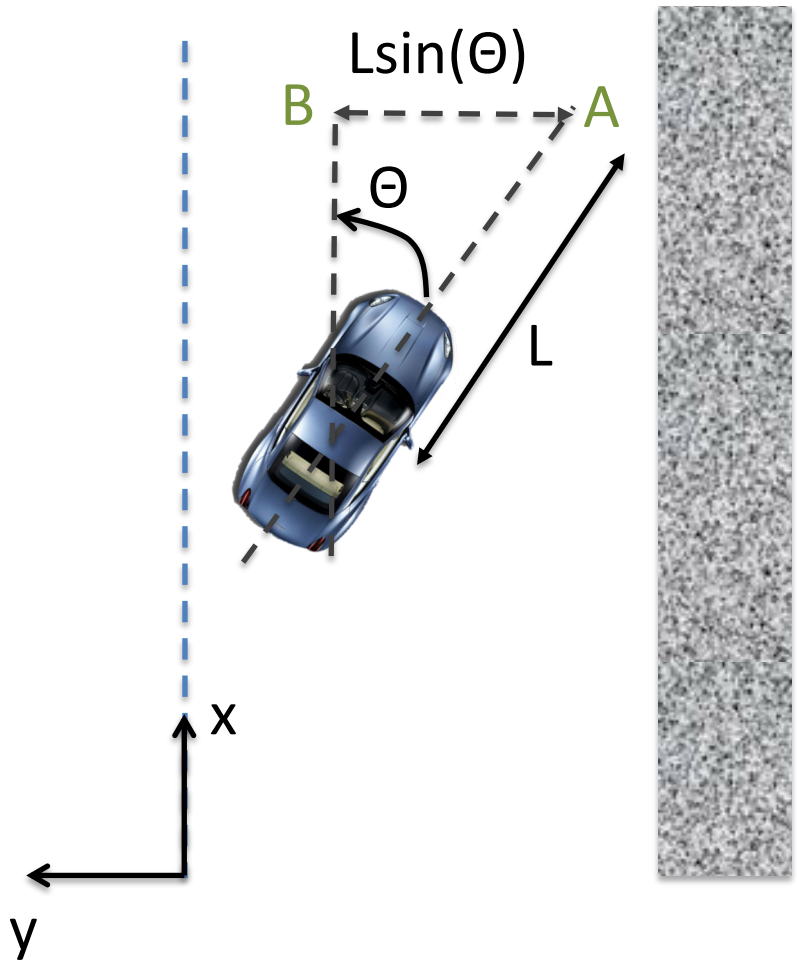
1) keep the car driving along the centerline,

$$y = 0$$

2) parallel to the walls.

$$\Theta = 0$$

PID control: control objectives



Control objective:

1) keep the car driving along the centerline,

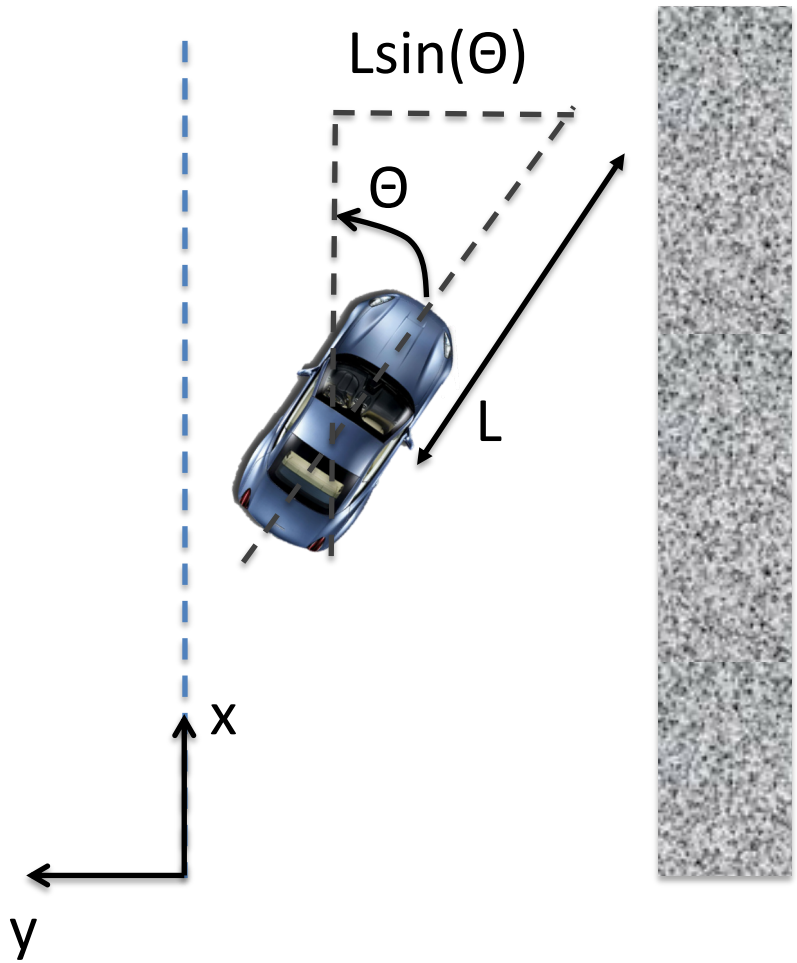
$$y = 0$$

2) After driving L meters, it is still on the centerline:

Horizontal distance after driving L meters

$$L\sin(\Theta) = 0$$

PID control: control inputs



Control input:
Steering angle Θ

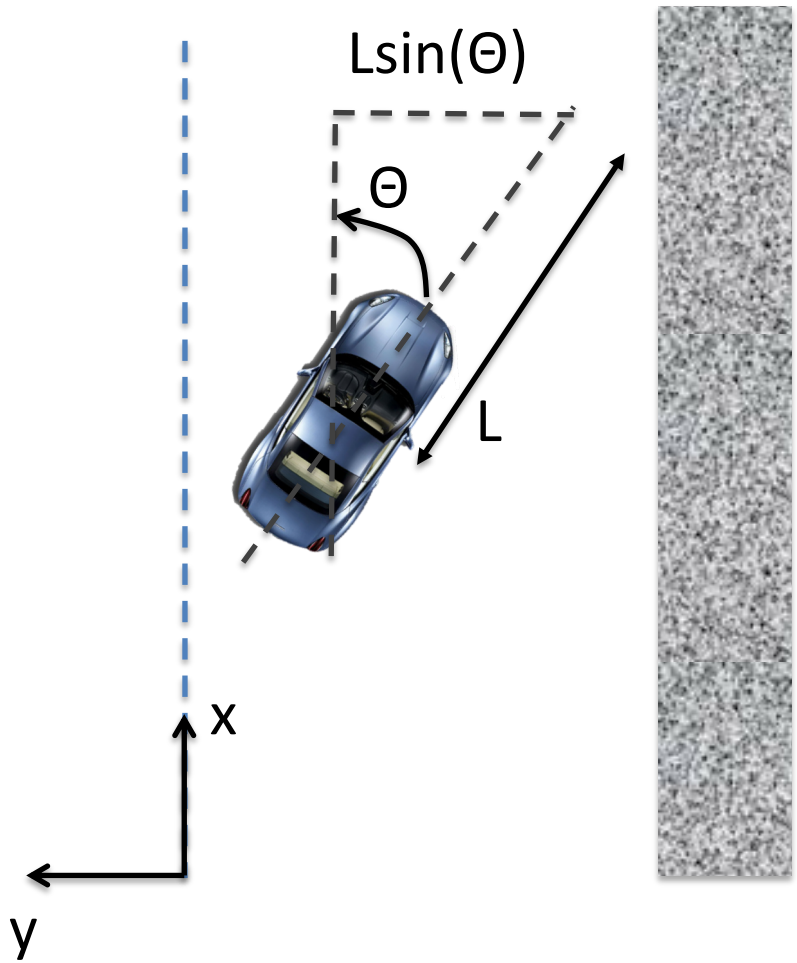
We will hold the velocity constant.

How do we control the steering angle to keep

$$y = 0, L\sin(\Theta) = 0$$

as much as possible?

PID control: error term

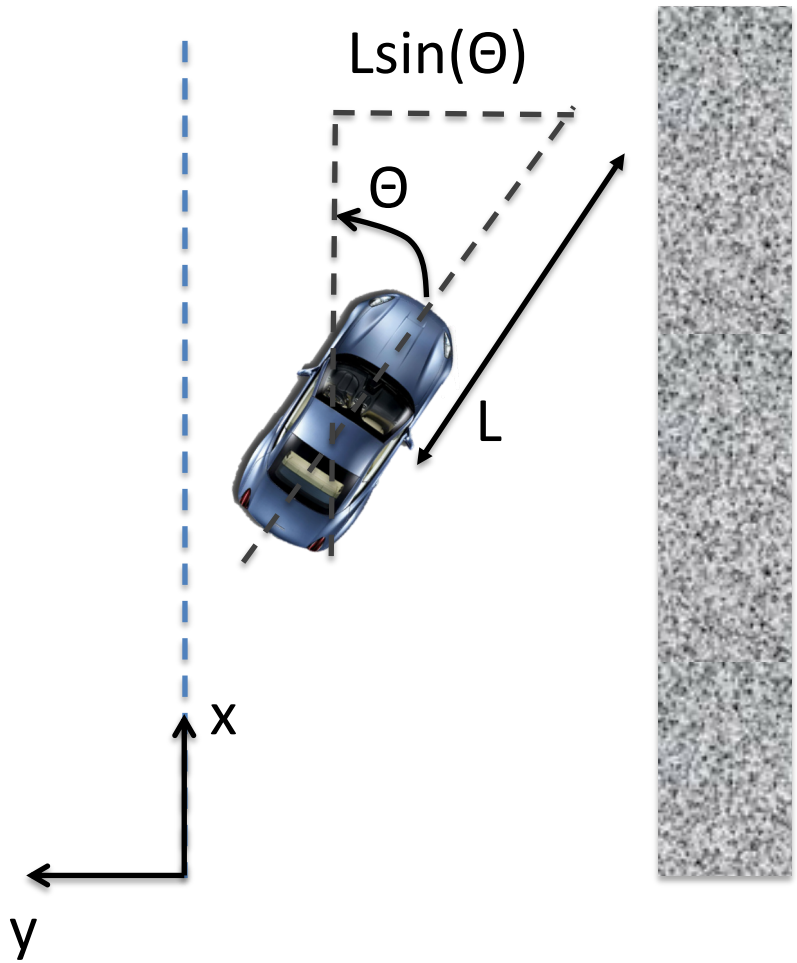


Want both y and Lsin(Θ) to be zero

→ Error term e(t) = -(y + Lsin(Θ))

We'll see why we added a minus sign

PID control: computing input



When y > 0, car is to the left of centerline

→ Want to steer right: Θ < 0

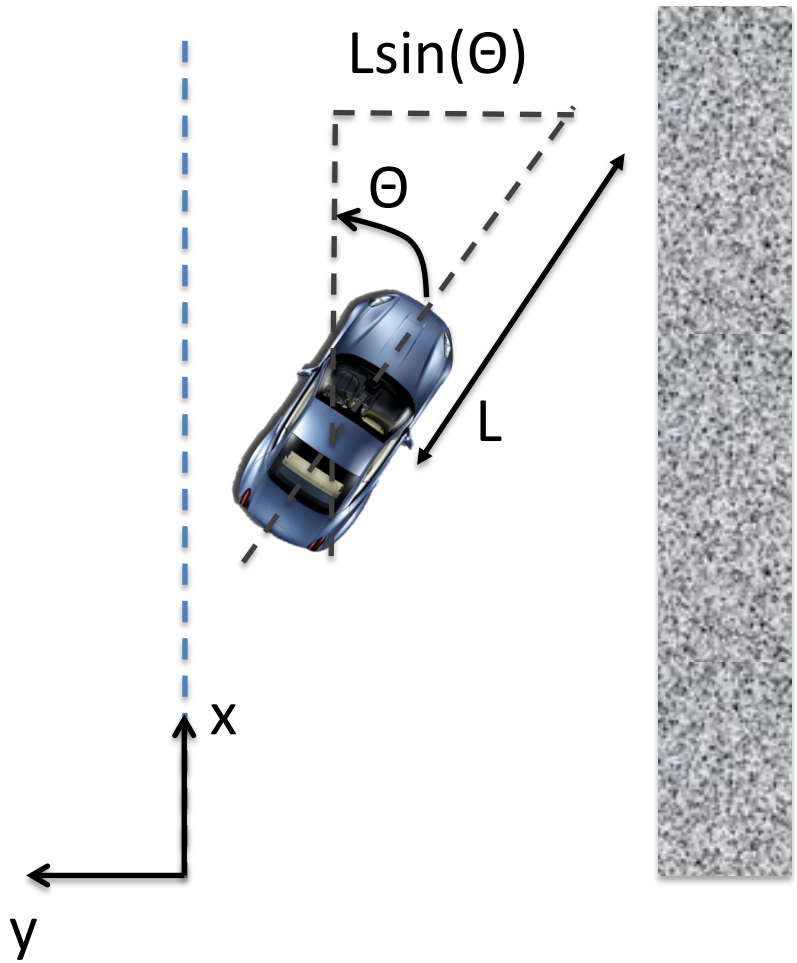
When Lsin(Θ) > 0, we will be to the left of centerline in L meters

→ so want to steer right: Θ < 0

Set *desired* angle to be

$\Theta_d = K_p (-y - L\sin(\Theta))$

PID control: computing input



When y < 0, car is to the right of centerline

→ Want to steer left

→ Want Θ > 0

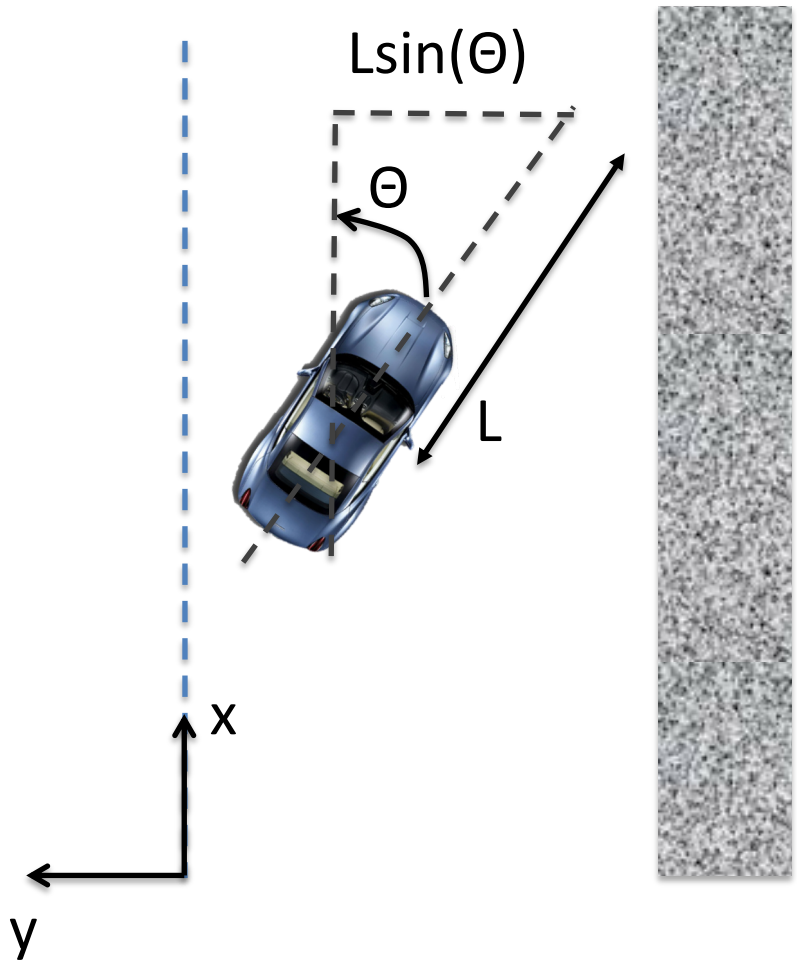
When Lsin(Θ) < 0, we will be to the right of centerline in L meters, so want to steer left

→Want Θ > 0

Consistent with previous requirement:

$\Theta_d = Kp\ (-y - Lsin(\Theta))$

PID control: Proportional control

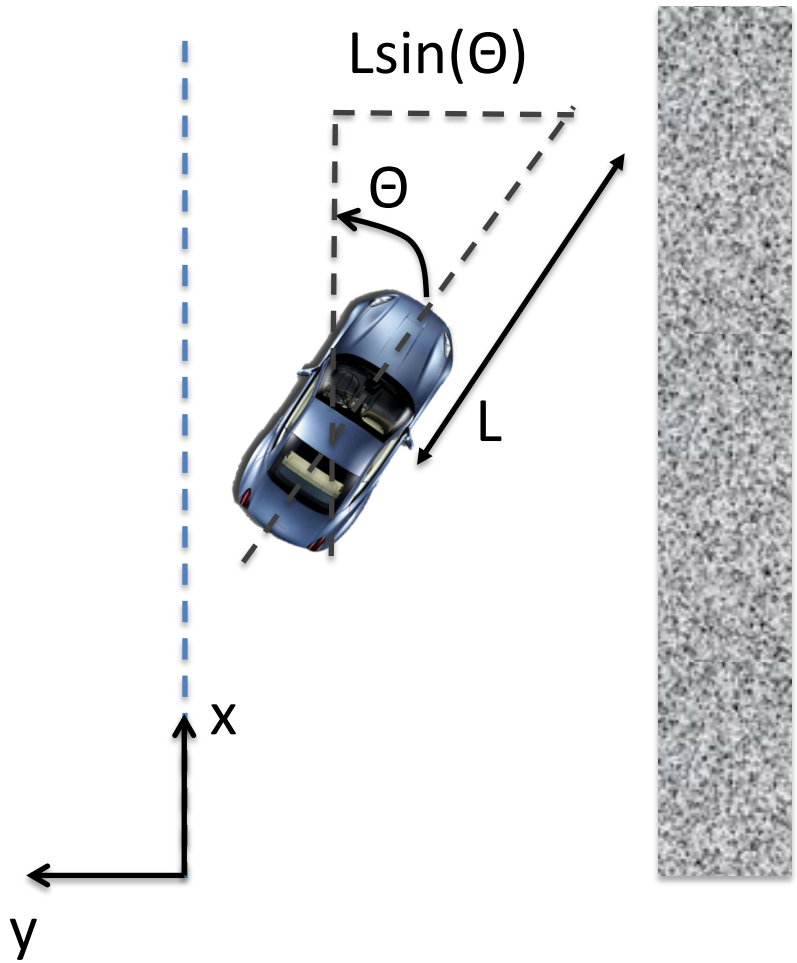


$\Theta_d = C\ K_p\ (-y - L\sin(\Theta)) = C\ K_p\ e(t)$

This is **P**roportional control.

The extra C constant is for scaling distances to angles.

PID control: Derivative control

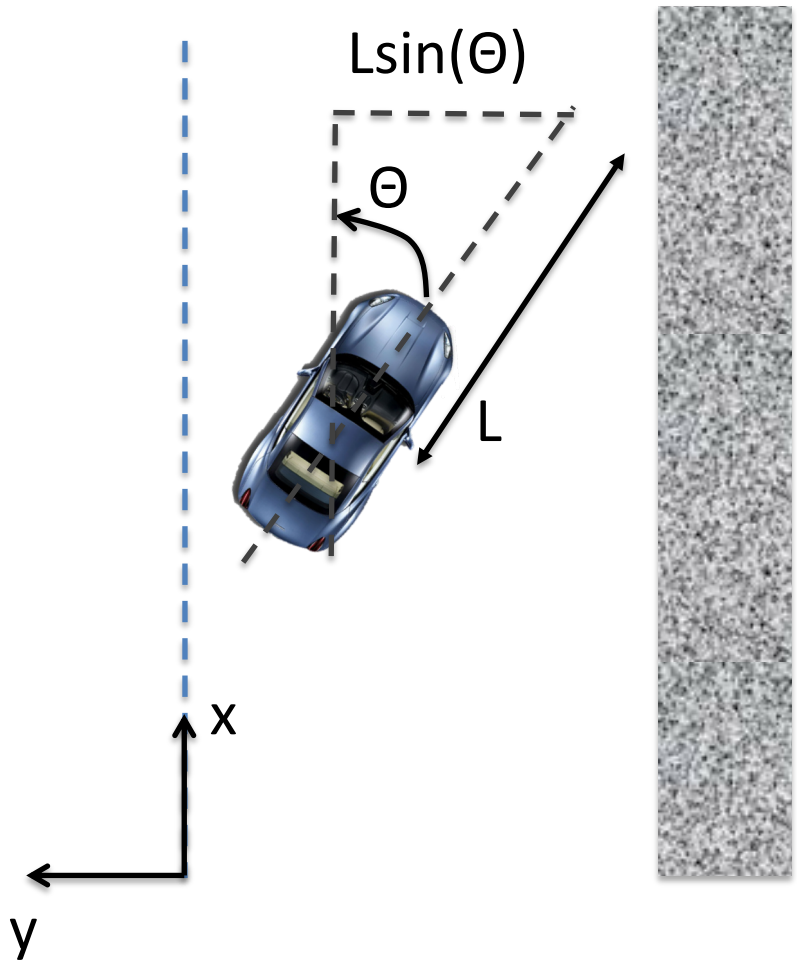


If error term is increasing quickly, we might want the controller to react quickly

→ Apply a *derivative gain*:

$$\Theta = K_p\, e(t) + K_d\, de(t)/dt$$

PID control: Integral control



Integral control is proportional to the *cumulative* error

$$\Theta = K_p\, e(t) + K_I\, E(t) + K_d\, de(t)/dt$$

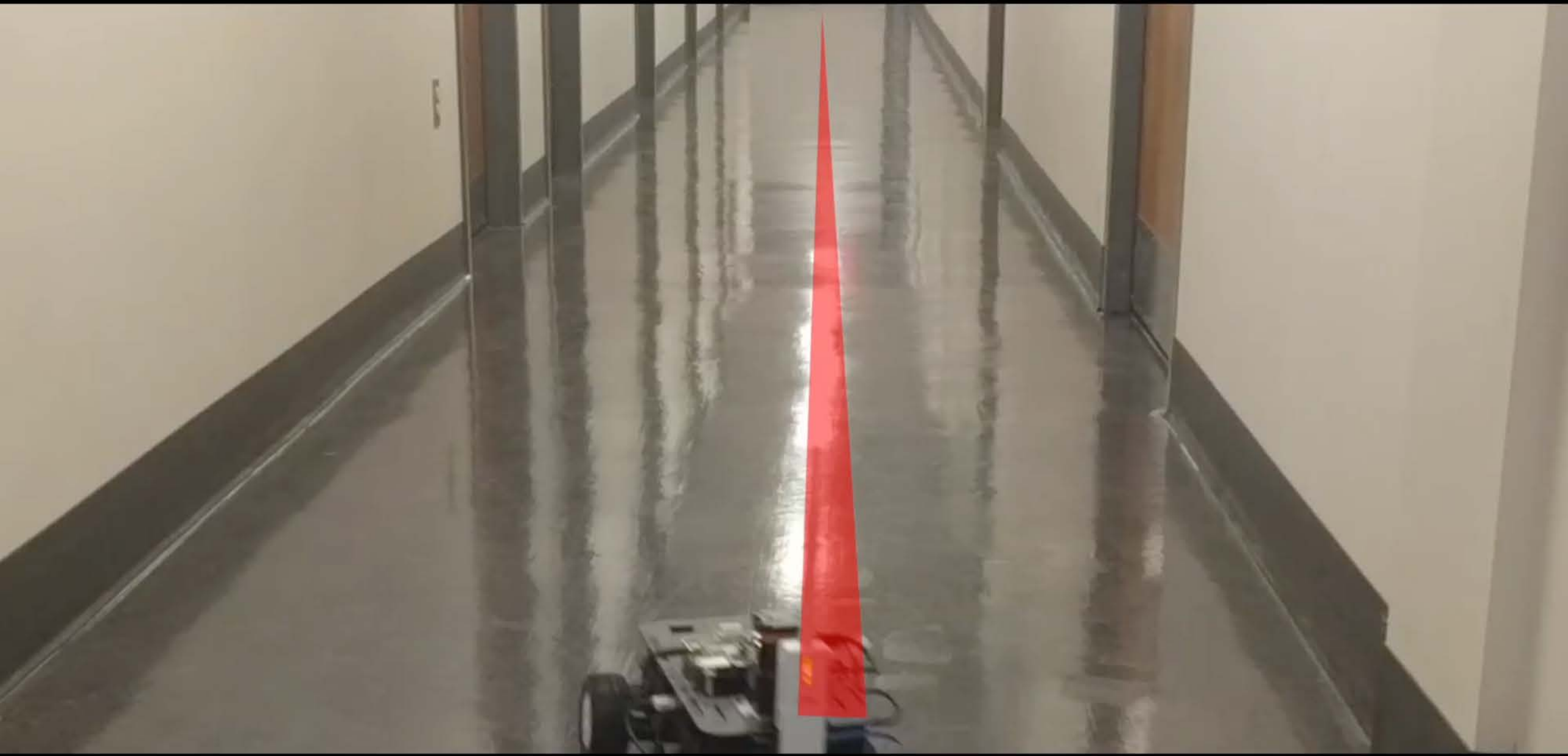
Where E(t) is the integral of the error up to time t (from a chosen reference time)

PID control: tuning the gains

- Default set of gains, determined empirically to work well for this car.
  - $K_p = 14$
  - $K_i = 0$
  - $K_d = 0.09$

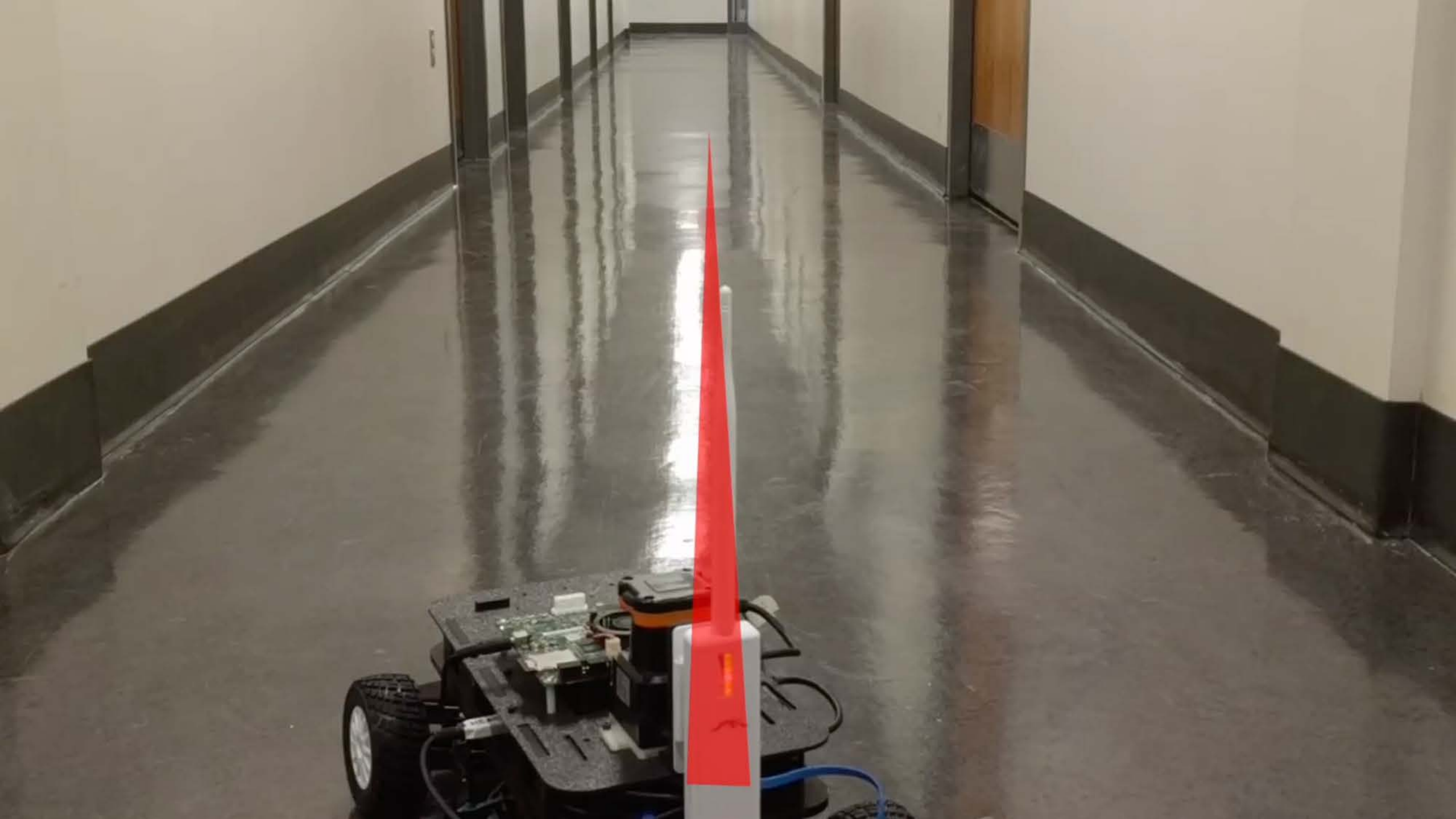
PID control: tuning the gains

- Reduce K_p → less responsive to error magnitude
  - $K_p = 5$
  - $K_i = 0$
  - $K_d = 0.09$



PID control: tuning the gains

- Include K_i → overly sensitive to accumulating error → over-correction
  - $K_p = 14$
  - $K_i = 2$
  - $K_d = 0.09$



Next Assignment..preview

- Implement Wall Following and demo in lab.
  - We will mark a tape `X' m away from the track boundary and evaluate if your PID controller can maintain the desired distance 'X' away from the wall.
  - Complete laps without crashing
- Implement Automatic Emergency Braking
  - If an obstacle appears directly in front of the car, and closer than a threshold distance during wall following, the car must come to an immediate stop.