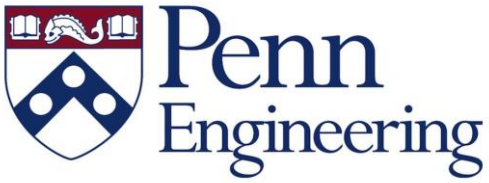
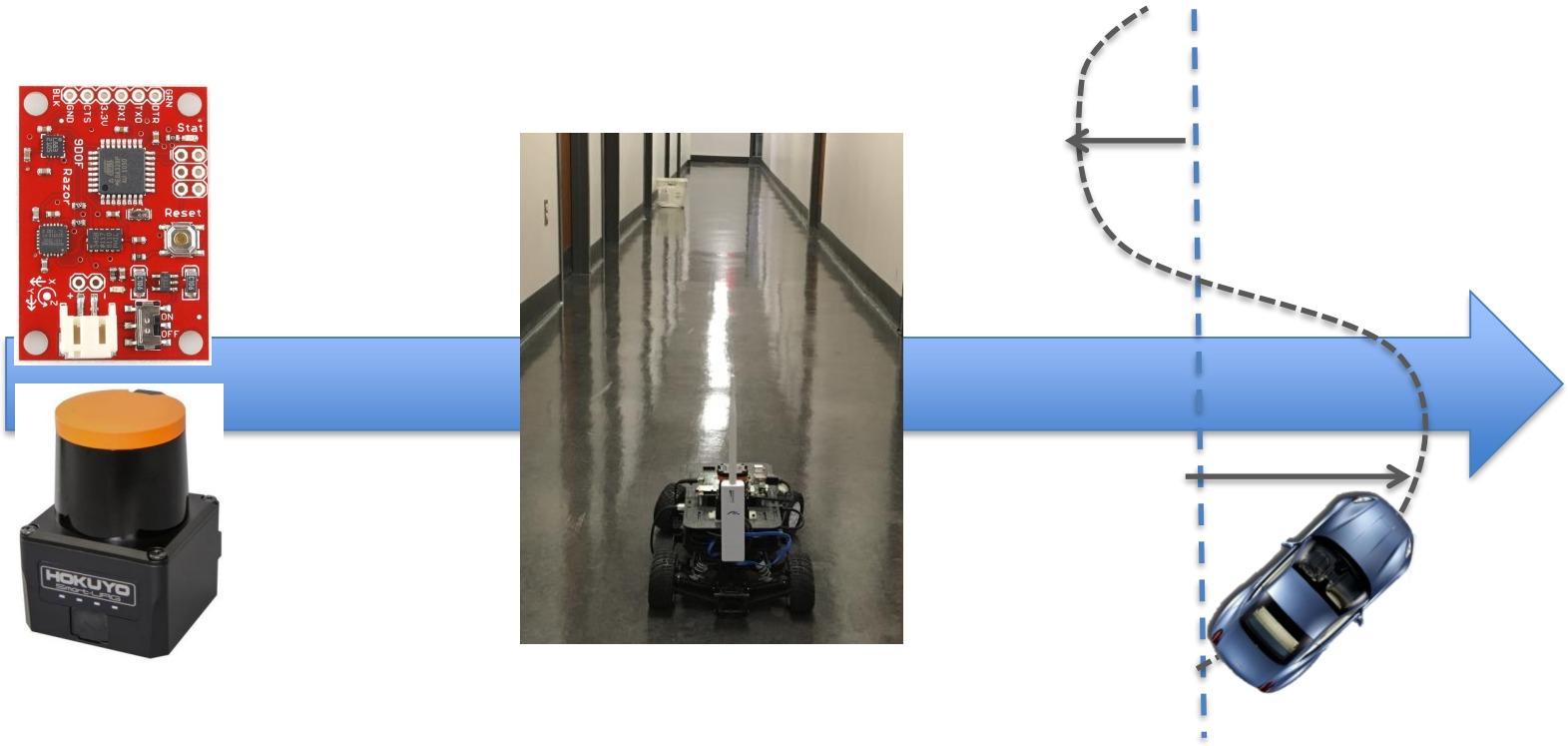
Perception, Planning and Control

F1/10th Racing



Perception, planning and control



IMU and LIDAR

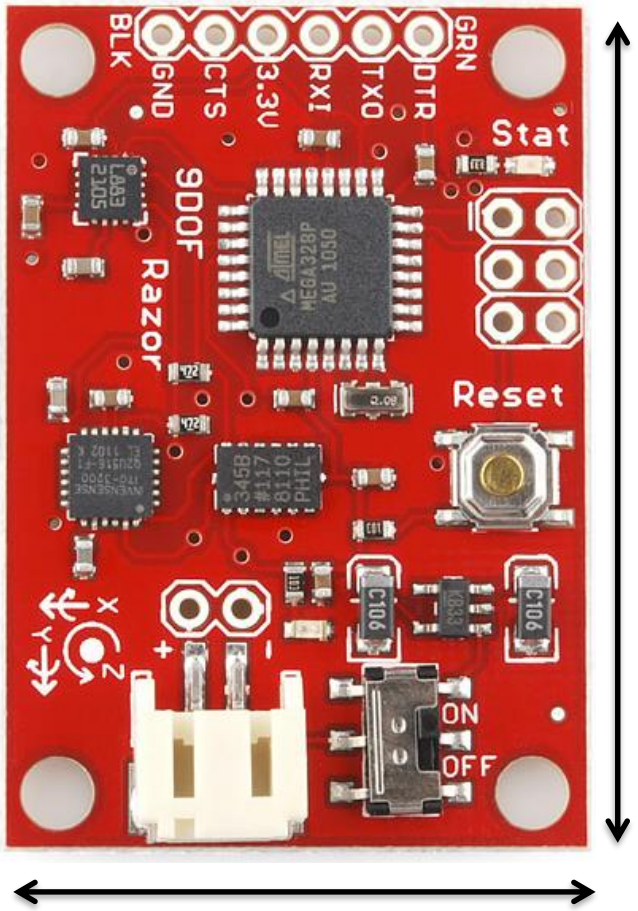
Localization

PID Control

Perception

Sensors: specifications, placement and data

Inertial Measurement Unit



41 mm

28 mm

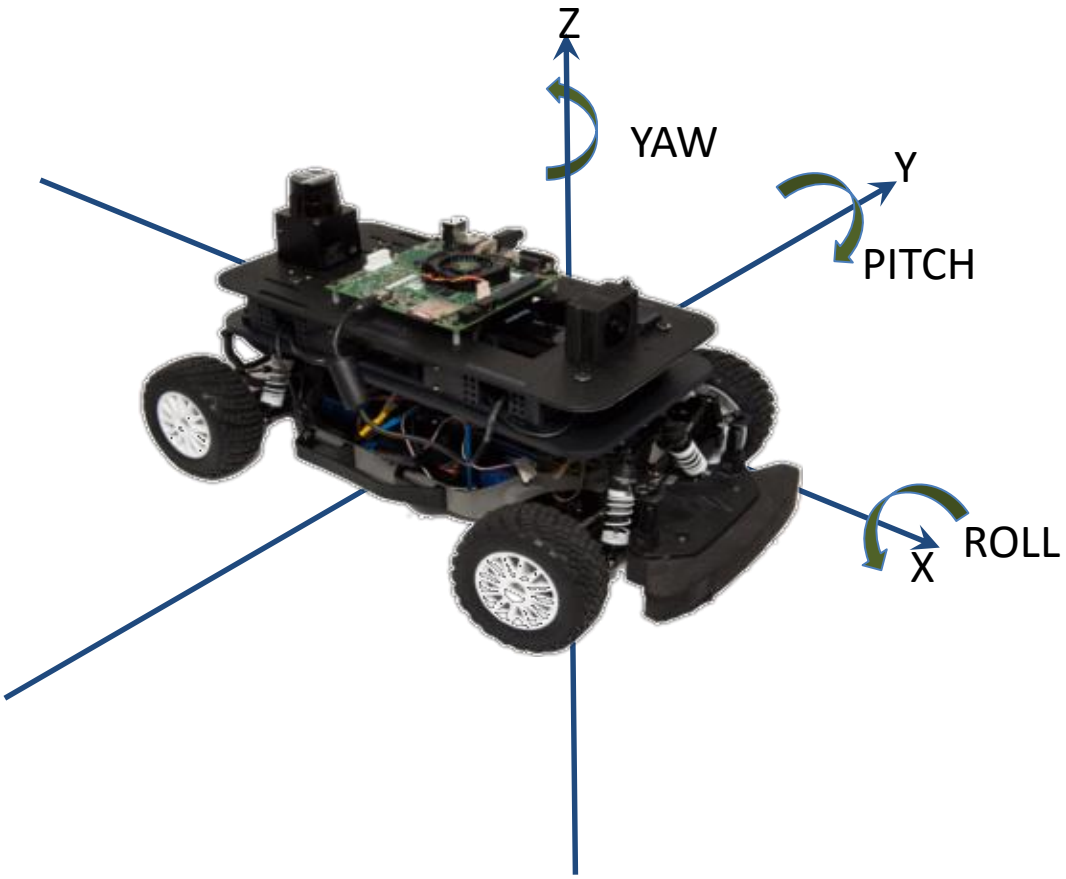
Razor IMU from Sparkfun

An **I**nertial **M**easurement **U**nit (IMU) provides acceleration, velocity and attitude measurements.

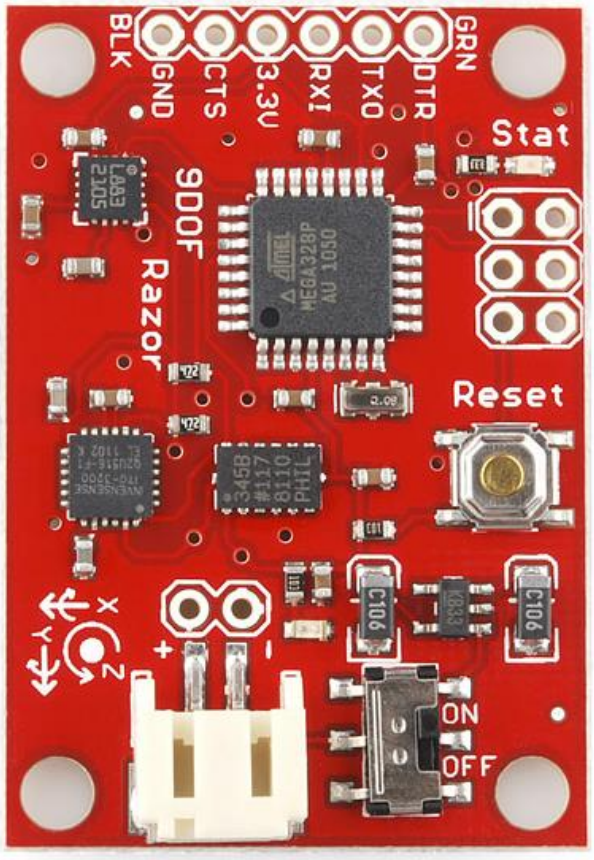
$$\dot{x} = v$$

$$\dot{v} = a$$

Inertial Measurement Unit: Attitude



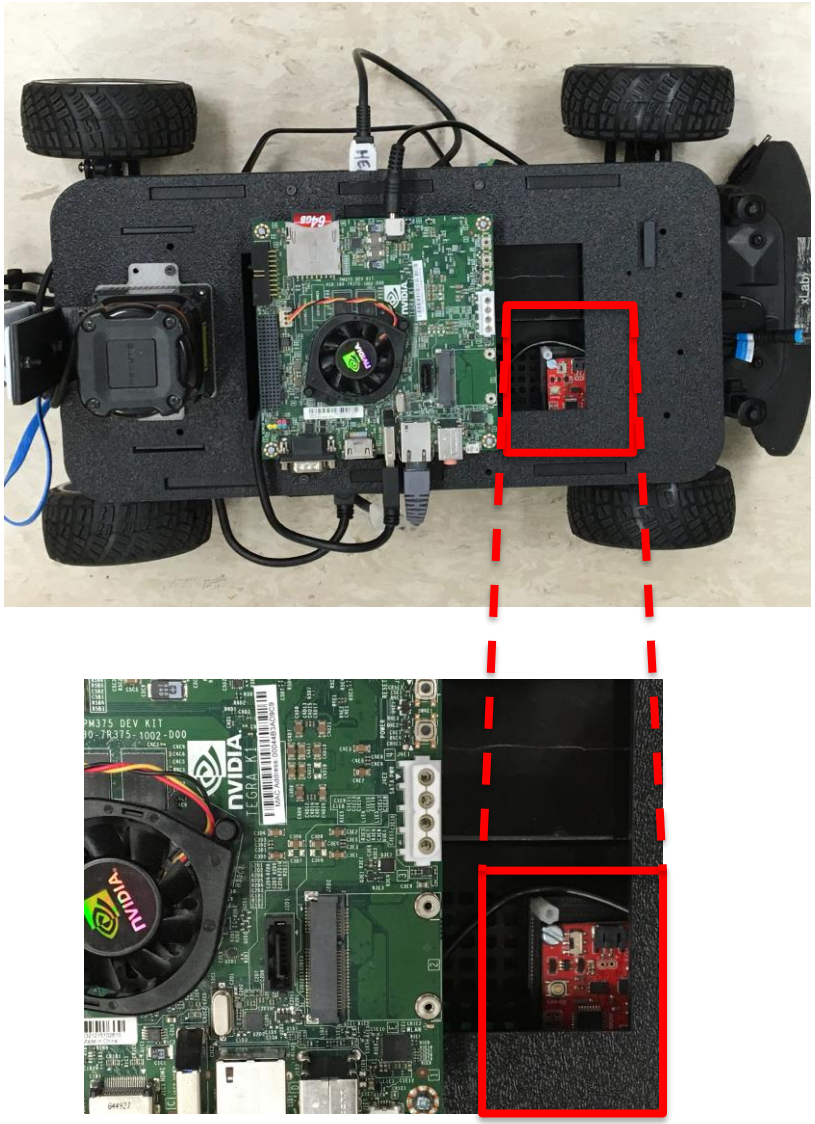
Inertial Measurement Unit



Razor IMU from Sparkfun

- Three gyroscopes measure the roll, pitch and yaw of the car.
- Three accelerometers measure the linear and rotational acceleration of the car.
- Three magnetometers

Inertial Measurement Unit: Placement



IMU: ROS message

File: **sensor_msgs/Imu.msg**
This is a message to hold data from an IMU (Inertial Measurement Unit)
#
Accelerations should be in m/s^2 (not in g's), and rotational velocity should be in rad/se
c
#
Header header

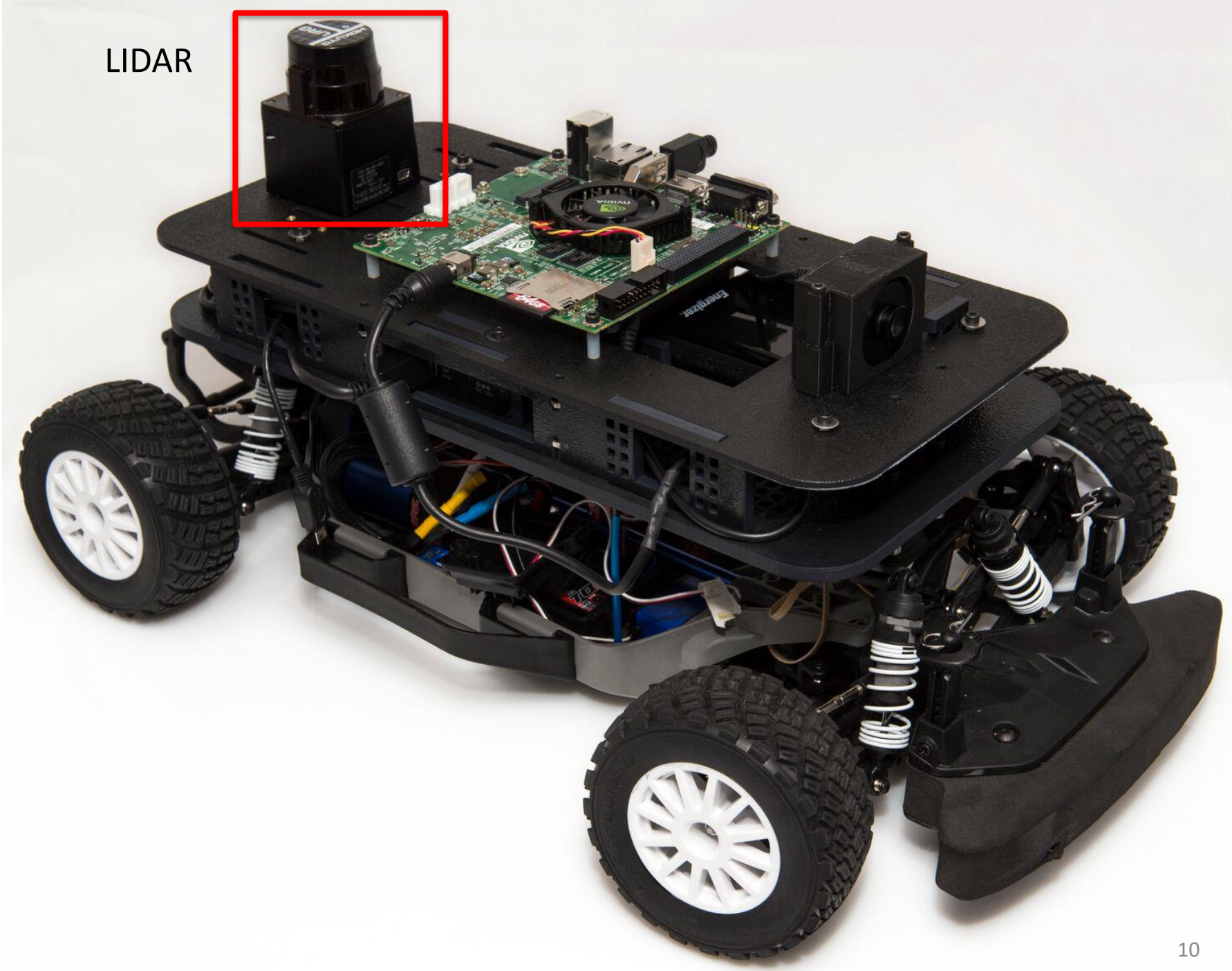
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance # Row major about x, y, z axes

geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance # Row major about x, y, z axes

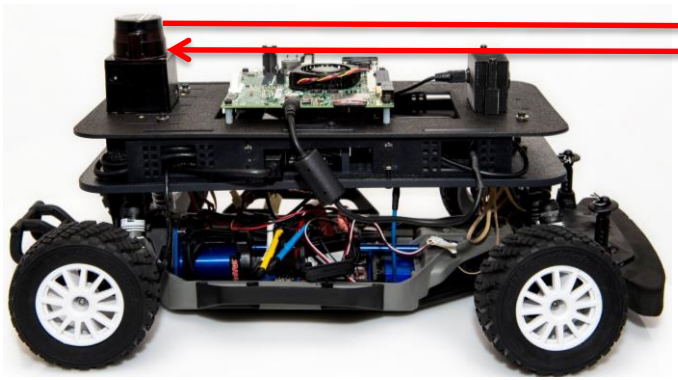
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance # Row major x, y z

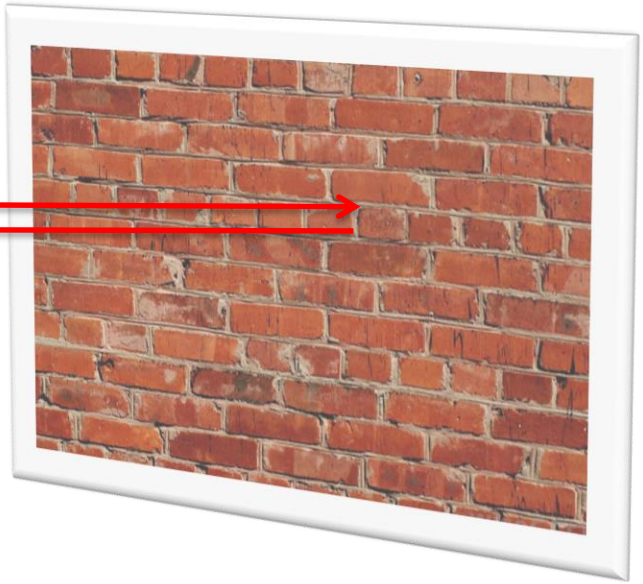
LIDAR

Depth information (how far are objects and obstacles around me)

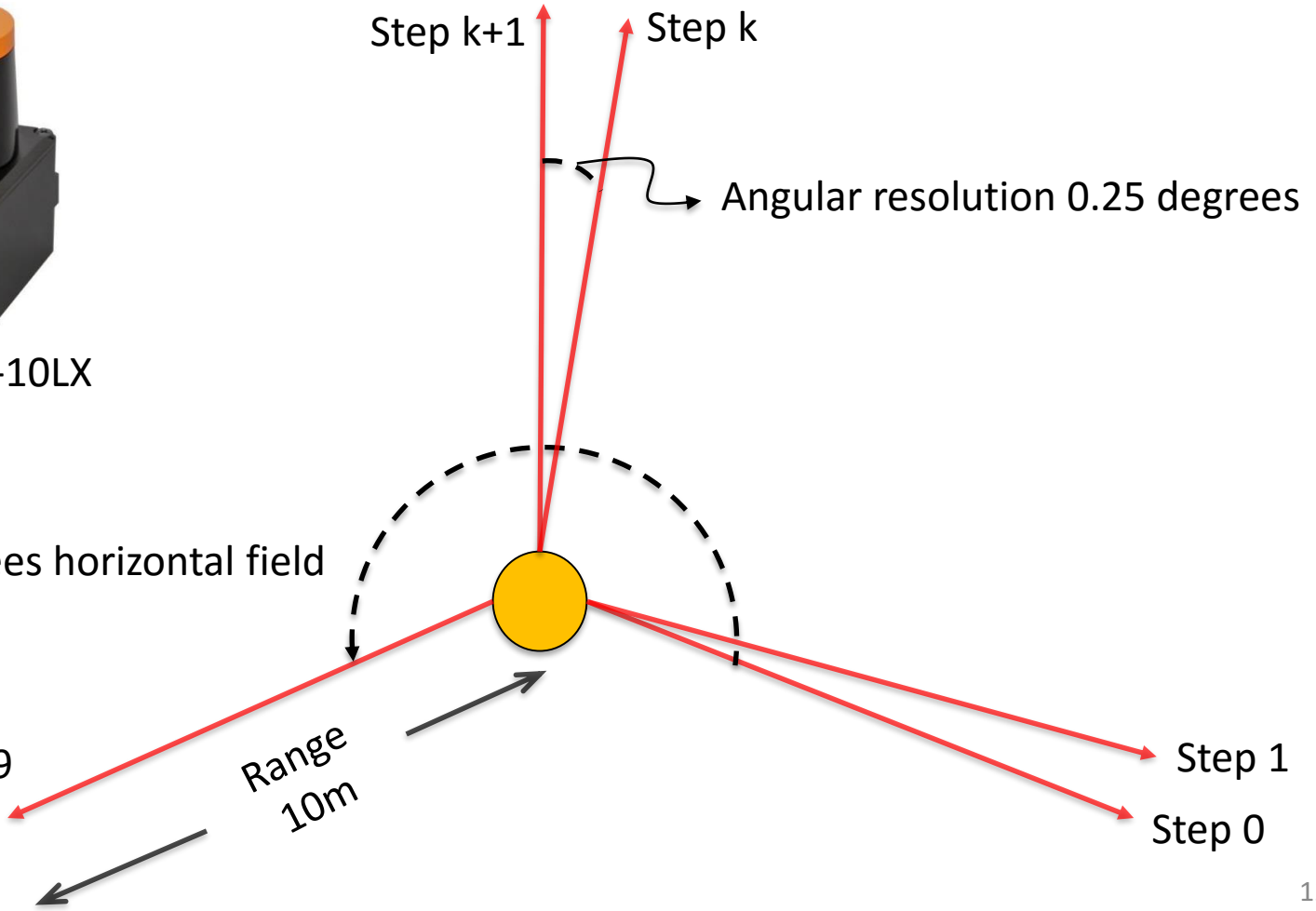

LIDAR

Distance to obstacle = (speed of light * time traveled)/2

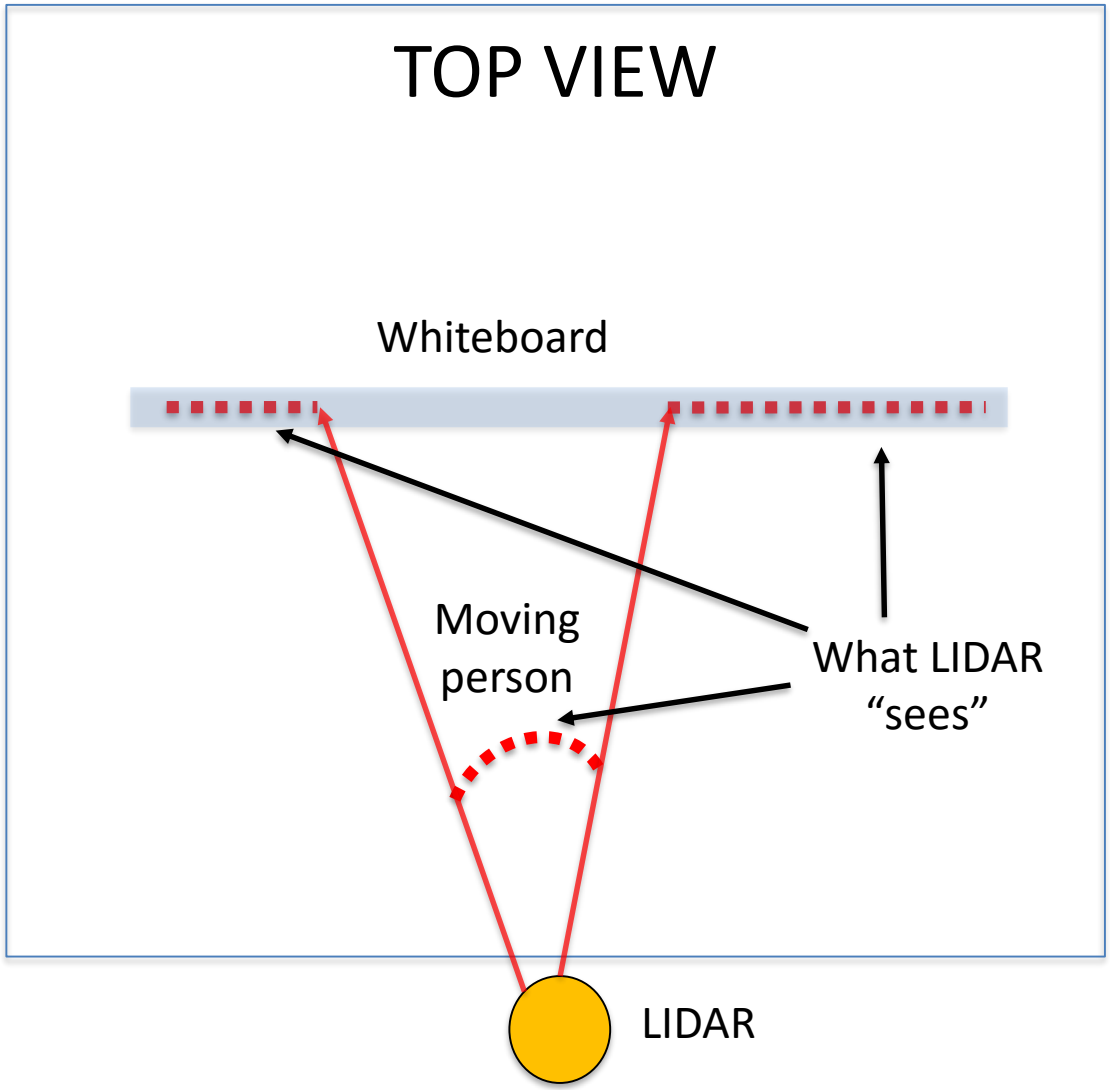




Hokuyo UST-10LX



LIDAR: a simple scan sequence

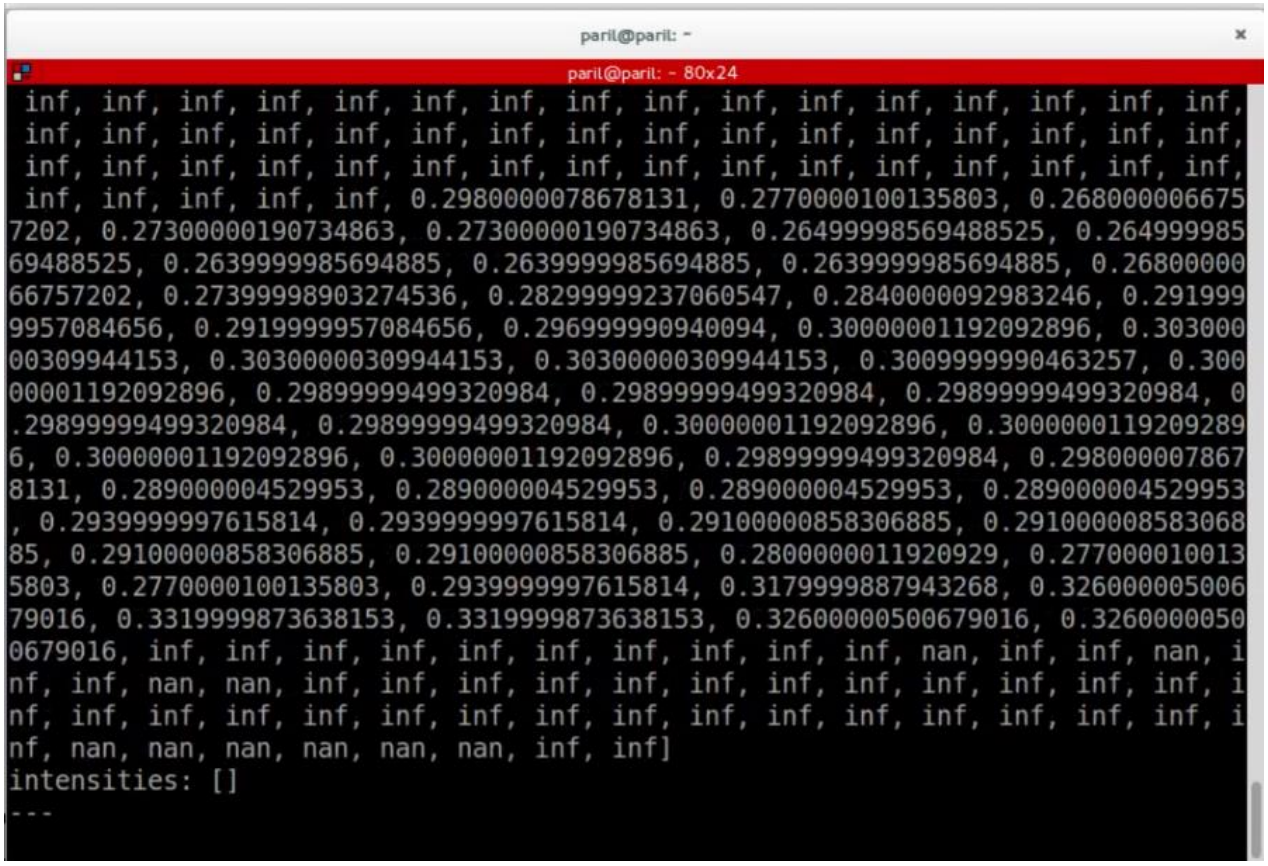


LIDAR: ROS message structure

```
# File: sensor_msgs/LaserScan.msg

Header header           # timestamp in the header is the acquisition time of
                         # the first ray in the scan.

float32 angle_min          # start angle of the scan [rad]
float32 angle_max          # end angle of the scan [rad]
float32 angle_increment    # angular distance between measurements [rad]

float32 time_increment     # time between measurements [seconds]
float32 scan_time          # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges           # range data [m] (Note: values < range_min or > range_max
                             # should be discarded)
float32[] intensities      # intensity data [device-specific units].
```

LIDAR: range data

Array ranges[1080]: ranges[i] is distance measurement of (i+1)st step.
Measurements beyond the min and max range (like inf) *should be discarded.*



Localization

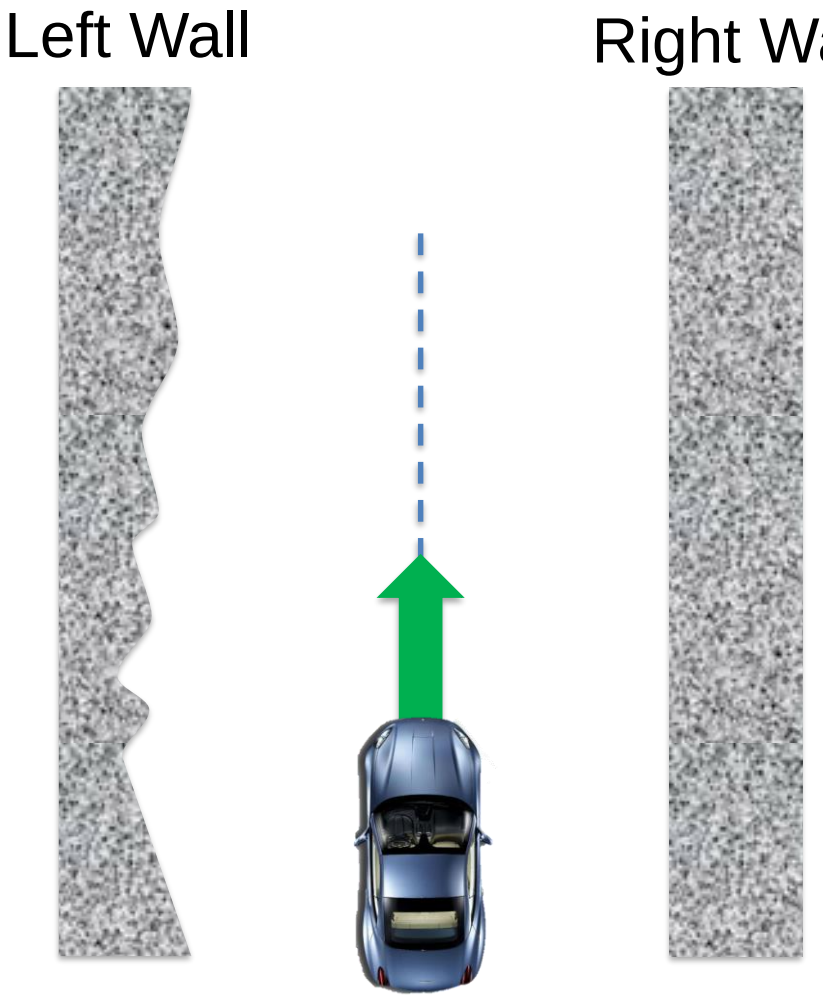
Know (where is) thyself:
Localization by LIDAR

Odometry with LIDAR

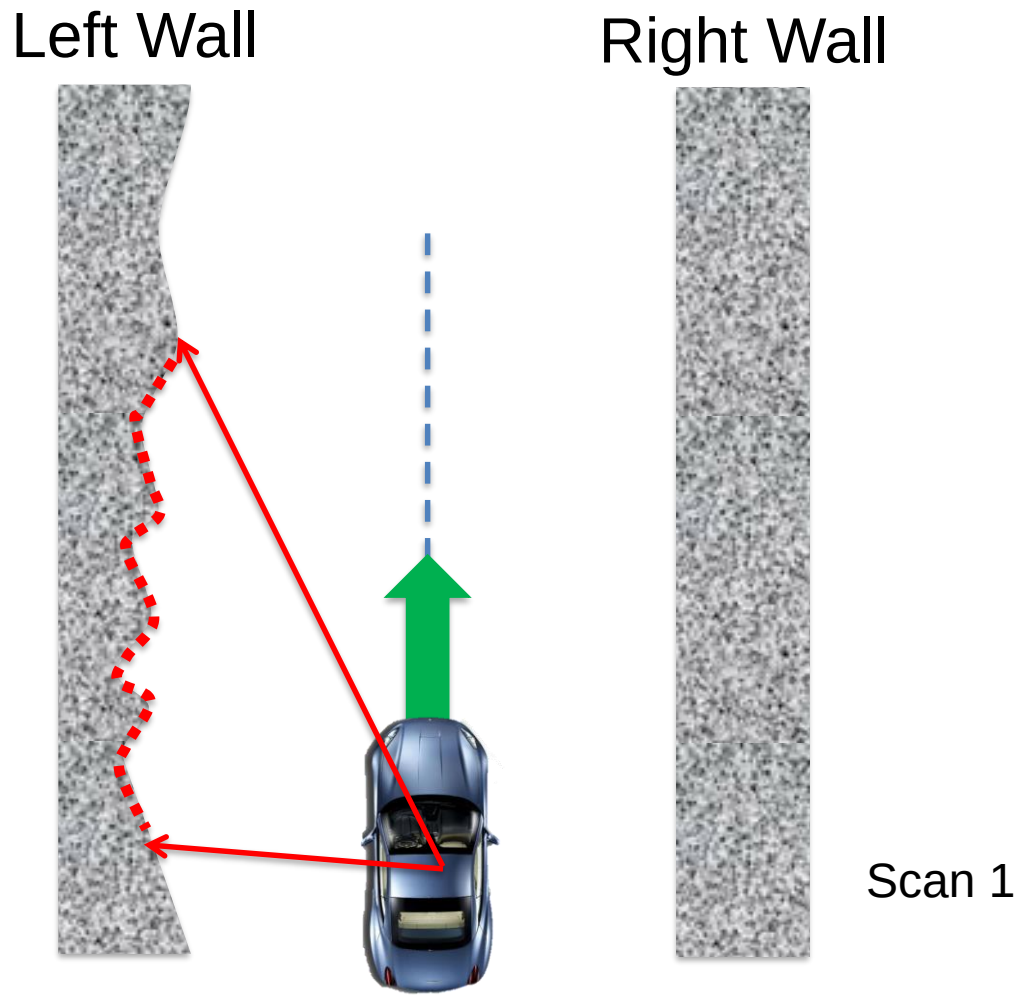
- Odometry = Estimation of position (x, y, z) and attitude (θ, φ, ψ)

- In what follows, will use “position” as short for “position and attitude”

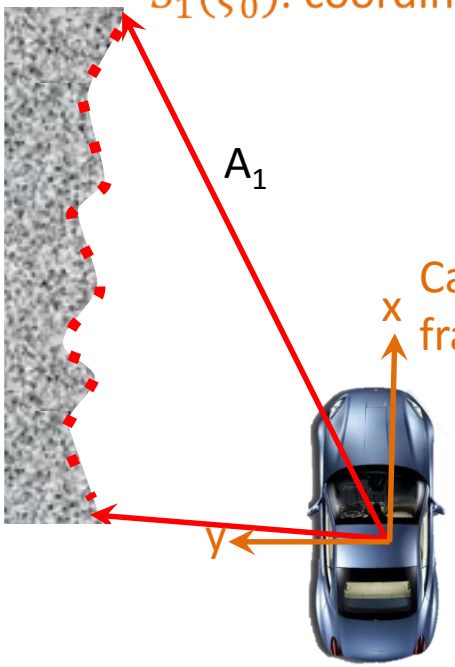
LIDAR for odometry: Scan matching

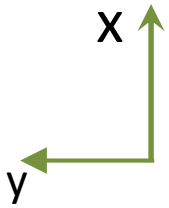


LIDAR for odometry: Scan matching



LIDAR for odometry: Scan 1

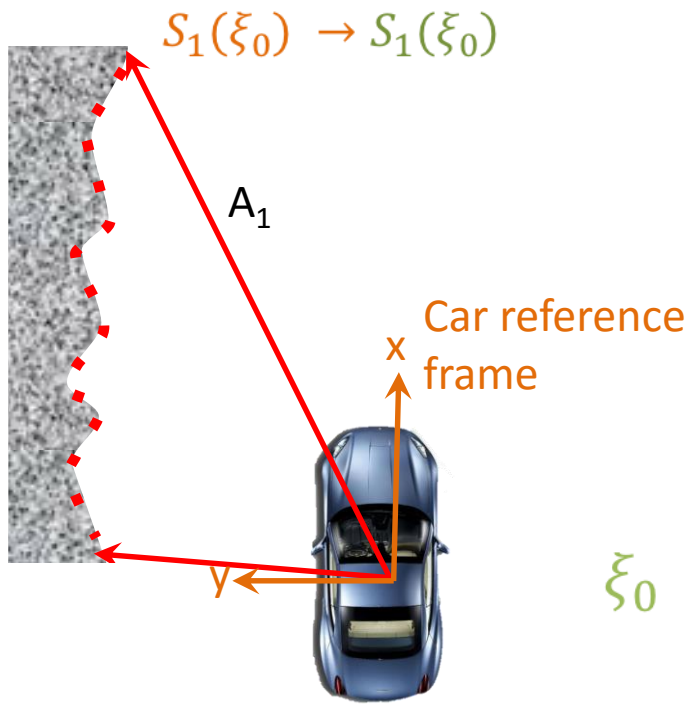


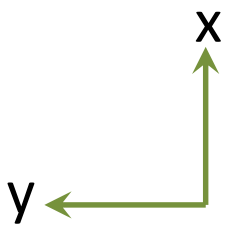


World reference frame

ξ_0 Initial position in world coordinates

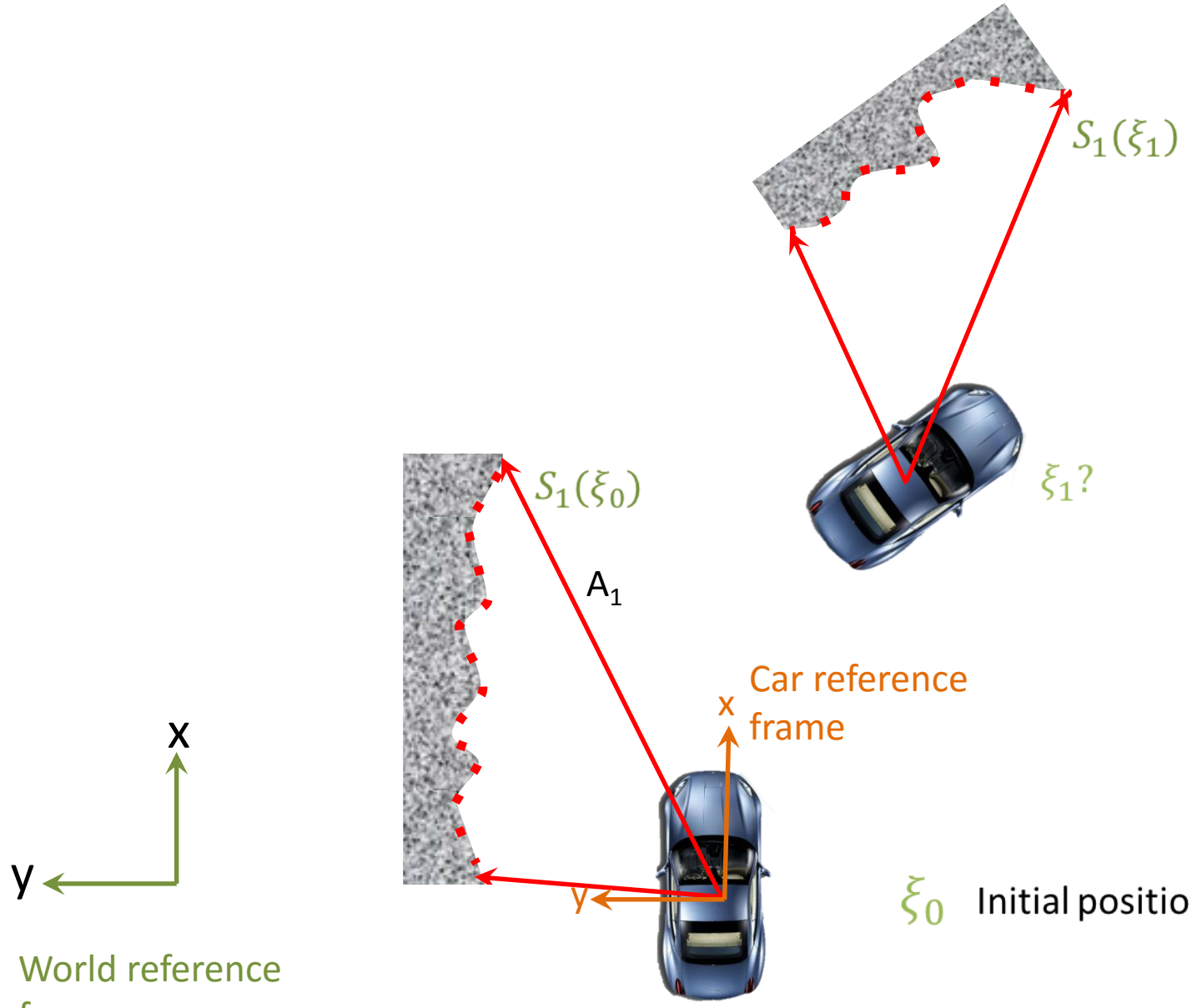
LIDAR for odometry: Scan 1



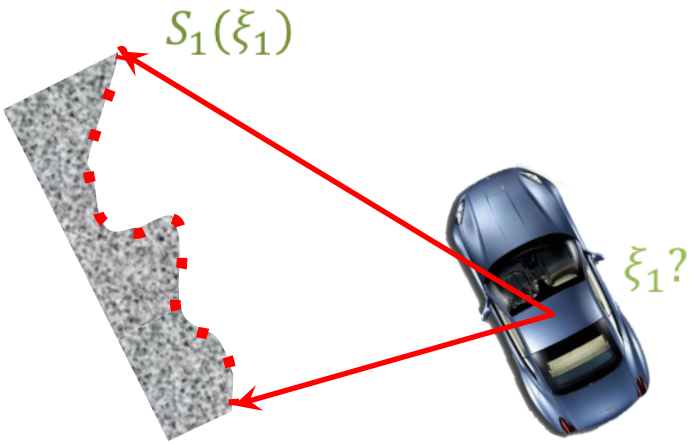


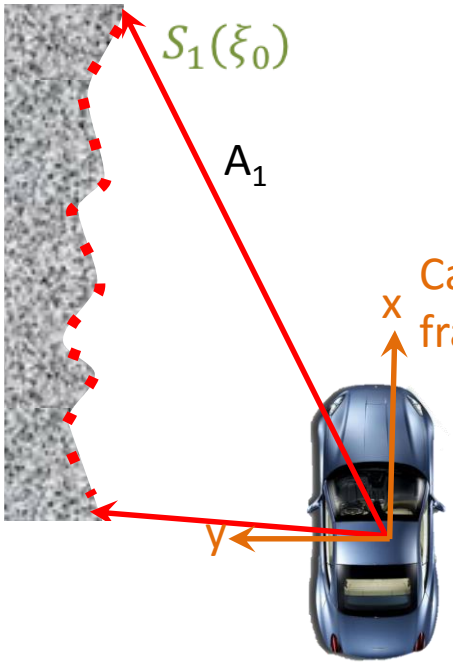
World reference frame

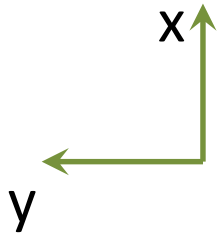
ξ_0 Initial position in world coordinates



ξ_0 Initial position in world coordinates

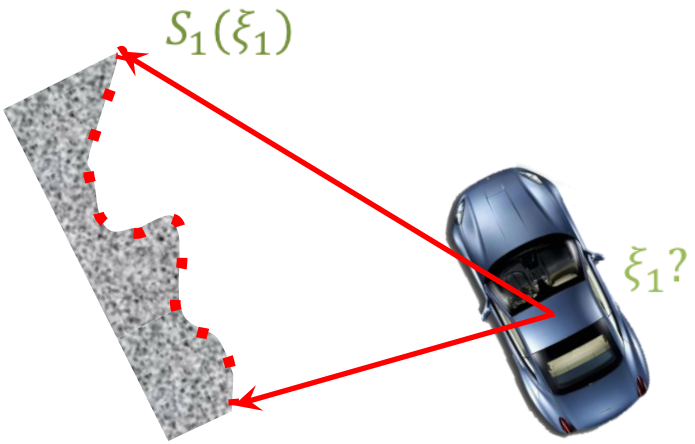


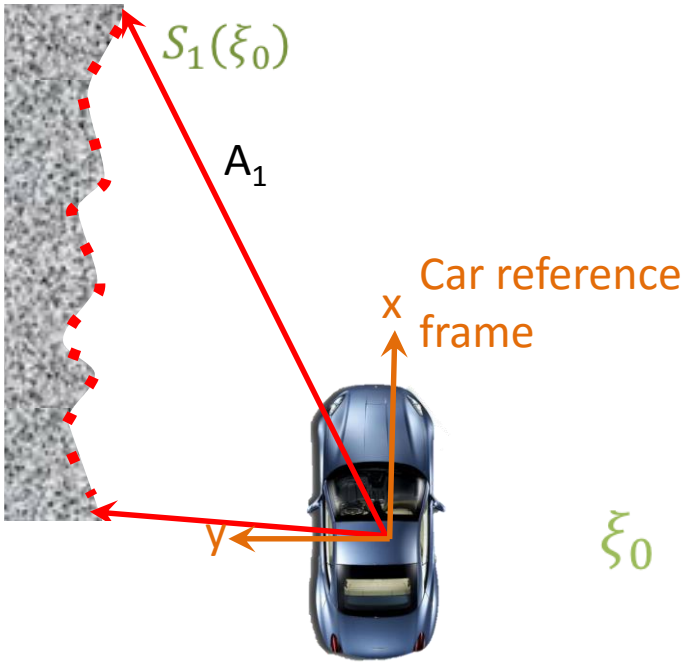


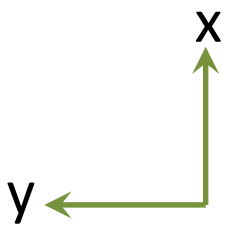


World reference frame

ξ_0 Initial position in world coordinates



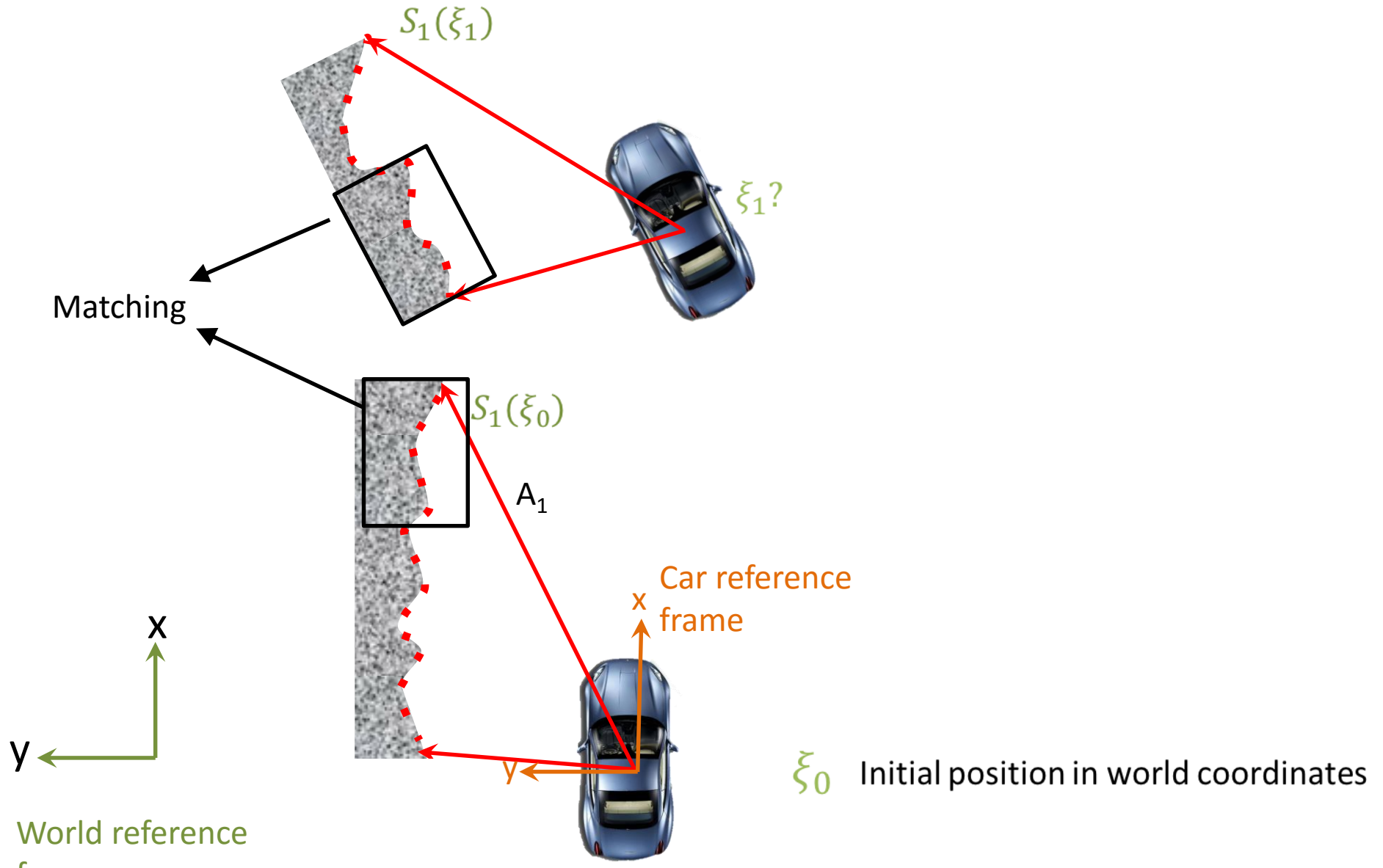


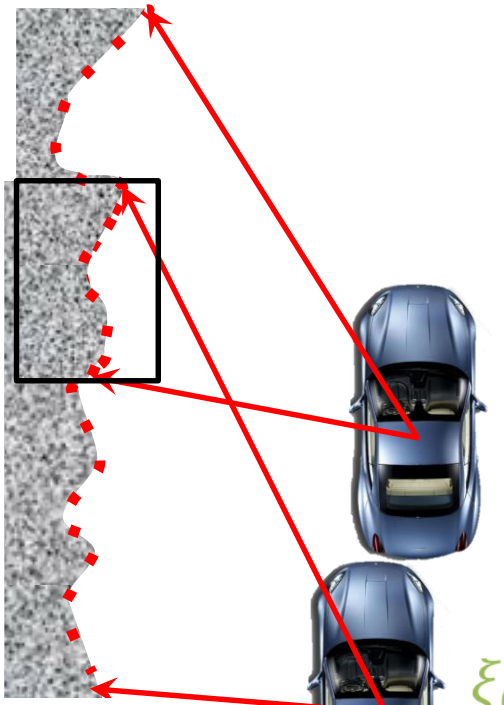


World reference frame

Premise: most likely car position at Scan 2 is the position that gives best overlap between the two scenes

ξ_0 Initial position in world coordinates


$S_1(\xi_1)$
ξ_1?
Matching
$S_1(\xi_0)$
A_1
Car reference frame
x
y
x
y
World reference frame
ξ_0 Initial position in world coordinates



ξ_1

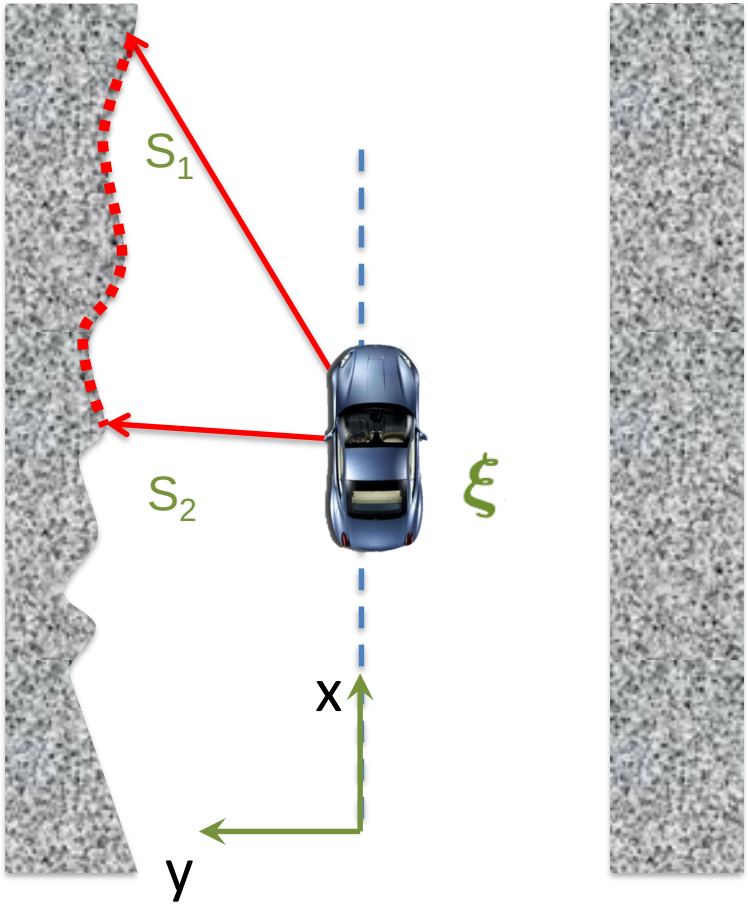
ξ_0 Initial position in world coordinates

x

y

World reference frame

Scan matching: optimization



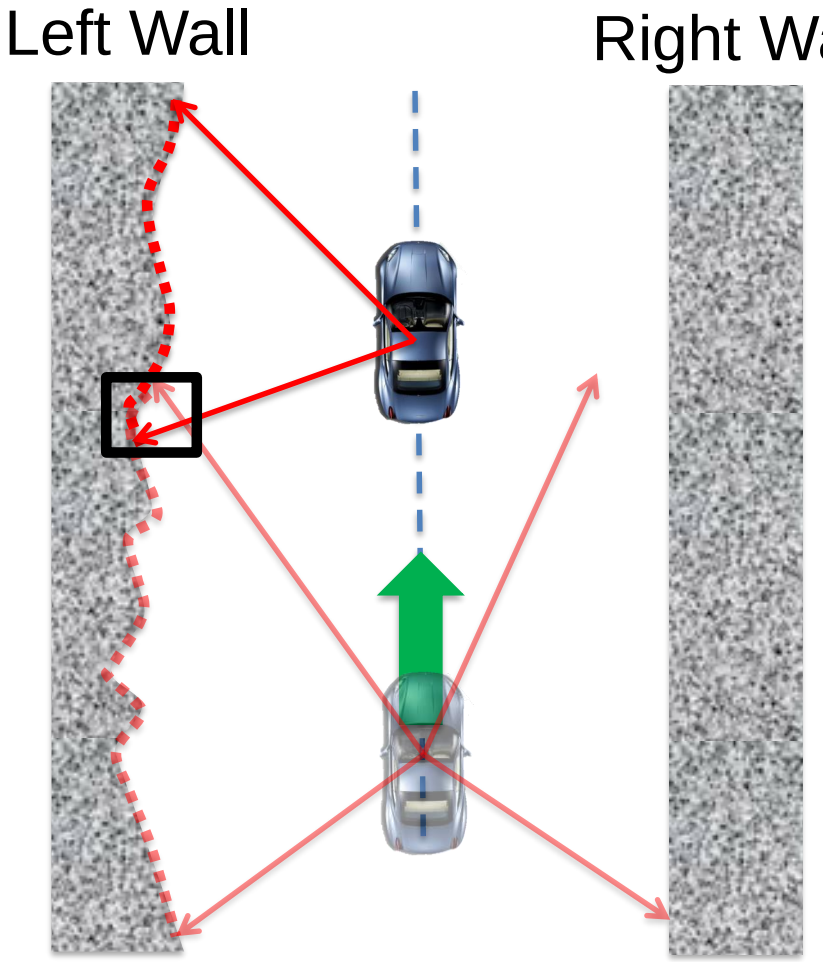
Impact coordinates of ith step in world frame

Total of n steps (n = 1084)

$$\boldsymbol{\xi}^* = \underset{\boldsymbol{\xi}}{\operatorname{argmin}} \sum_{i=1}^{n} \left[1 - M(\mathbf{S}_i(\boldsymbol{\xi}))\right]^2$$

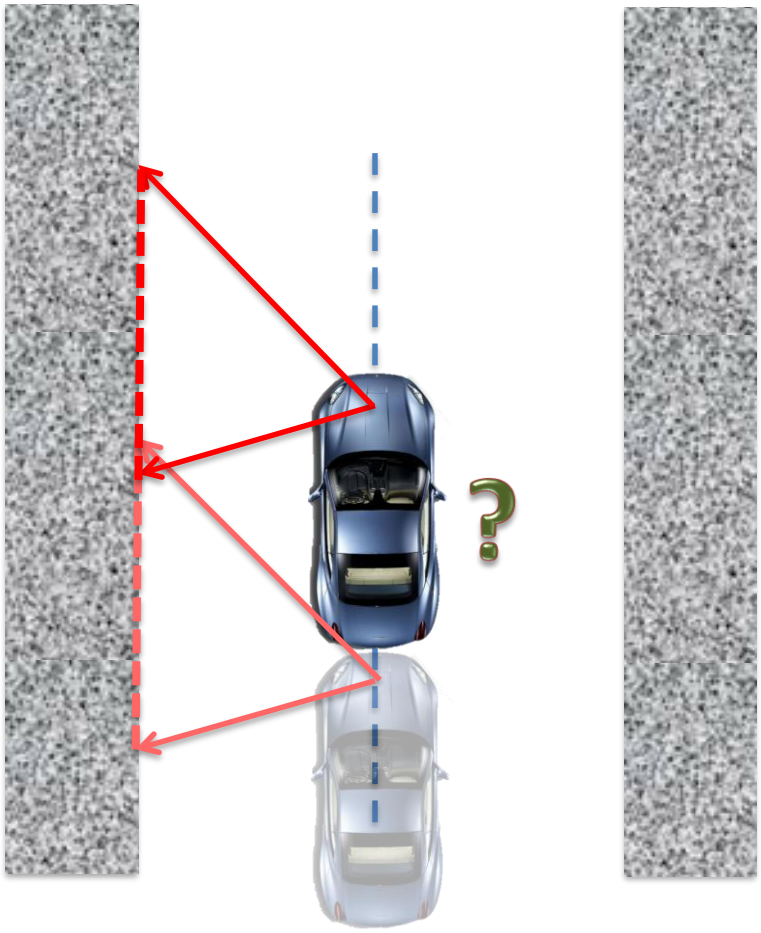
Measure of match

Scan matching: requirements



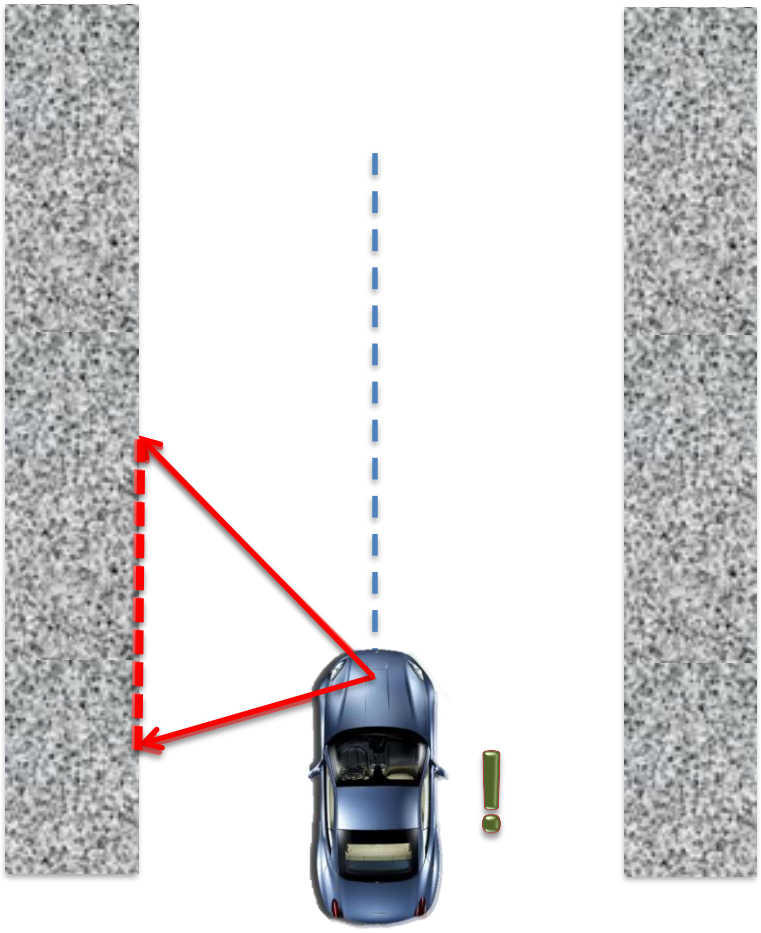
- Sufficiently fast scan rate
- A slow scan rate can lead to few matches between scans.
- Not really a risk for today's LIDARs

Scan matching : requirements



- Non-smooth, or heterogeneous, surfaces.
- Smooth surfaces all look the same to the matching algorithm.

Scan matching : requirements

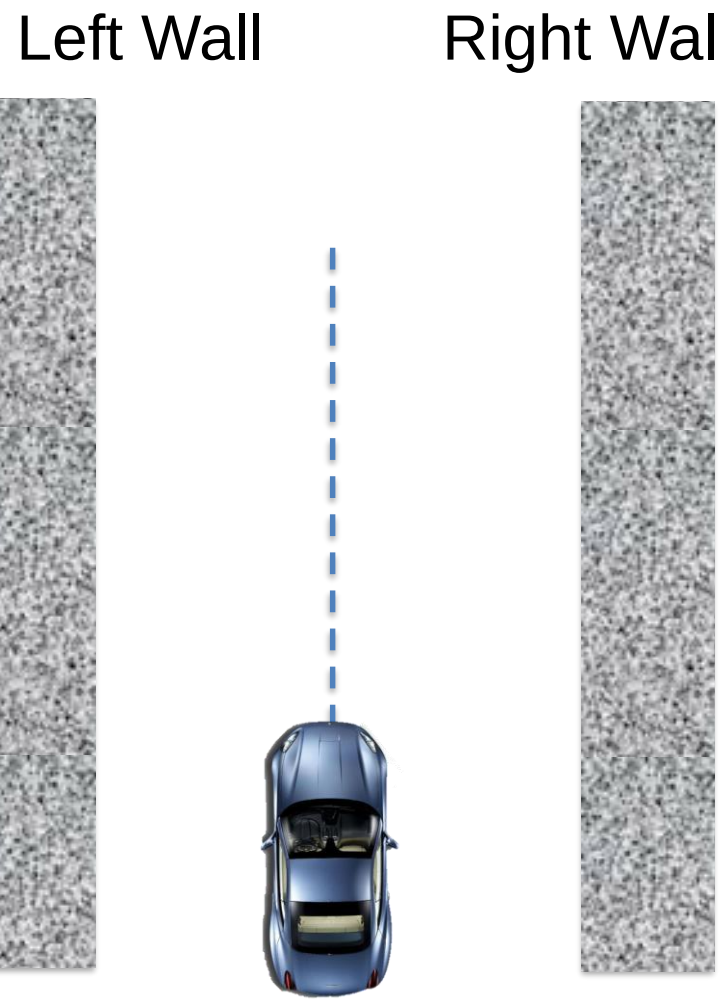


- Non-smooth, or heterogeneous, surfaces.
- Best match between scans is if car did not move.

Control

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

PID control: objectives

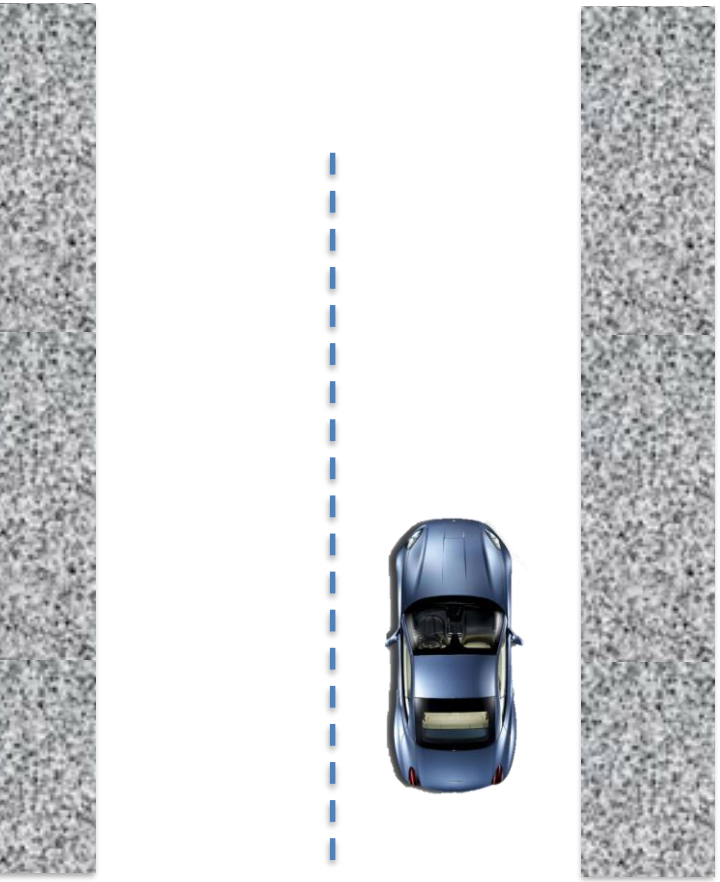


Control objective:

1) keep the car driving along the centerline,

2) parallel to the walls.

PID control: objectives

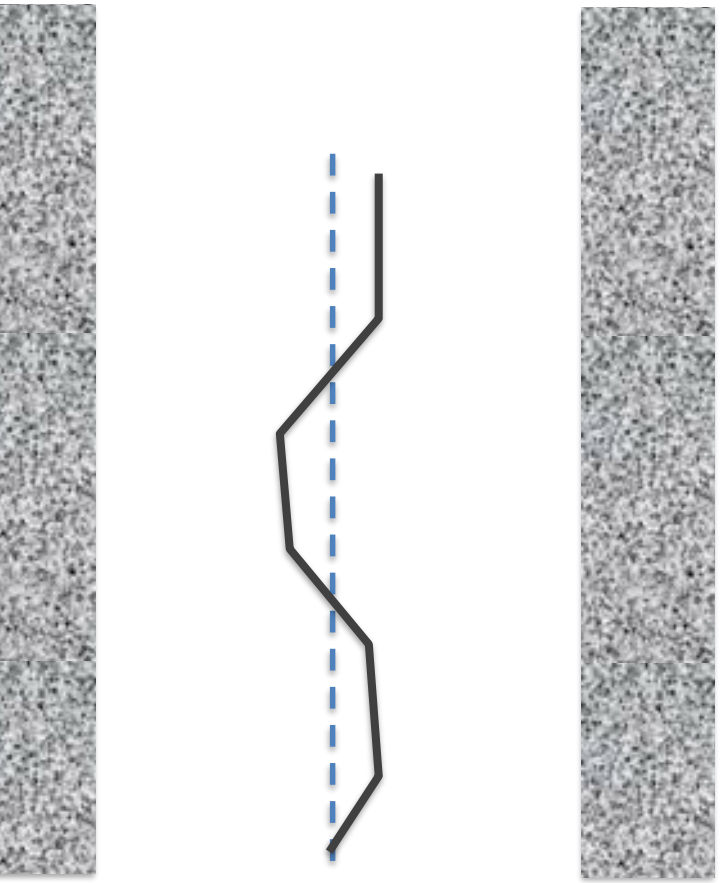


Control objective:

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

2) parallel to the walls.

PID control: objectives

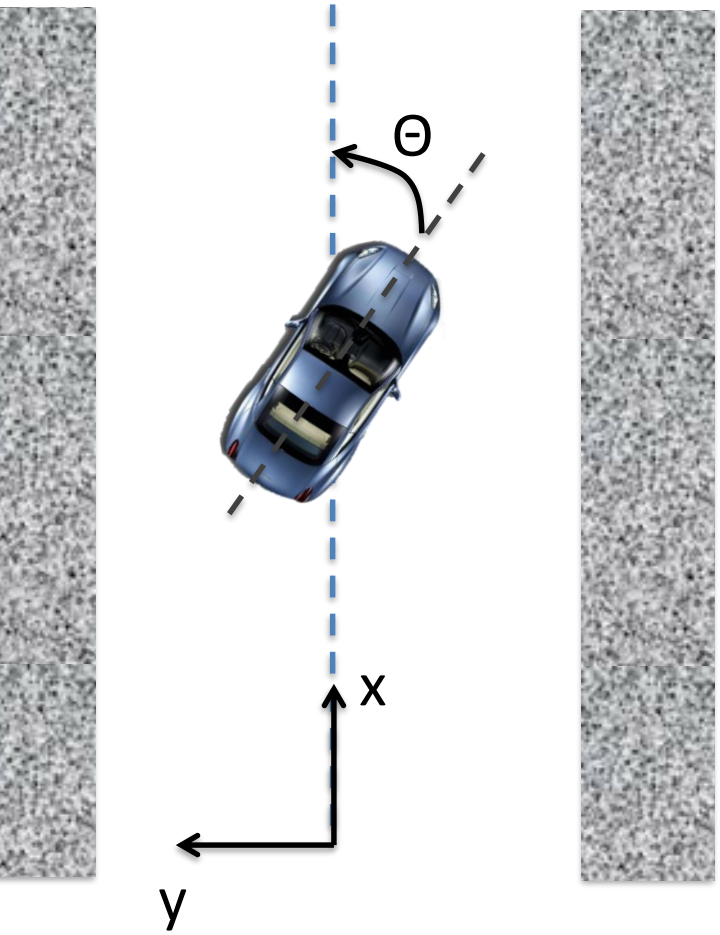


Control objective:

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

~~2) parallel to the walls.~~

PID control: control objectives



Control objective:

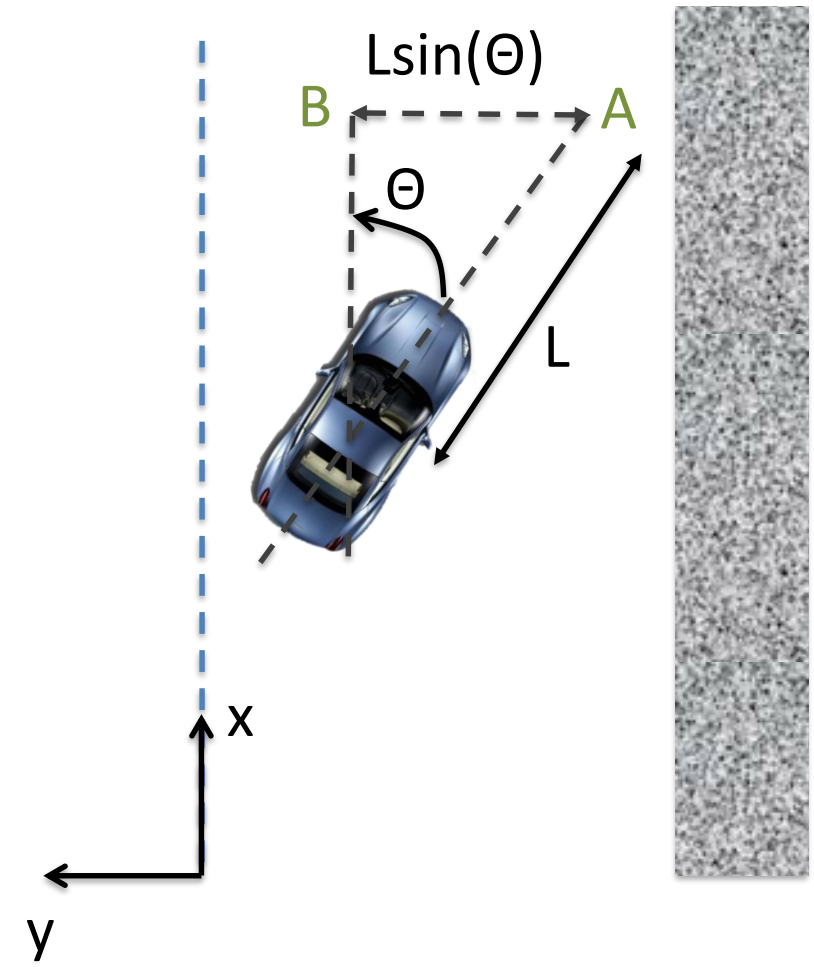
1) keep the car driving along the centerline,

$$y = 0$$

2) parallel to the walls.

$$\Theta = 0$$

PID control: control objectives



Control objective:

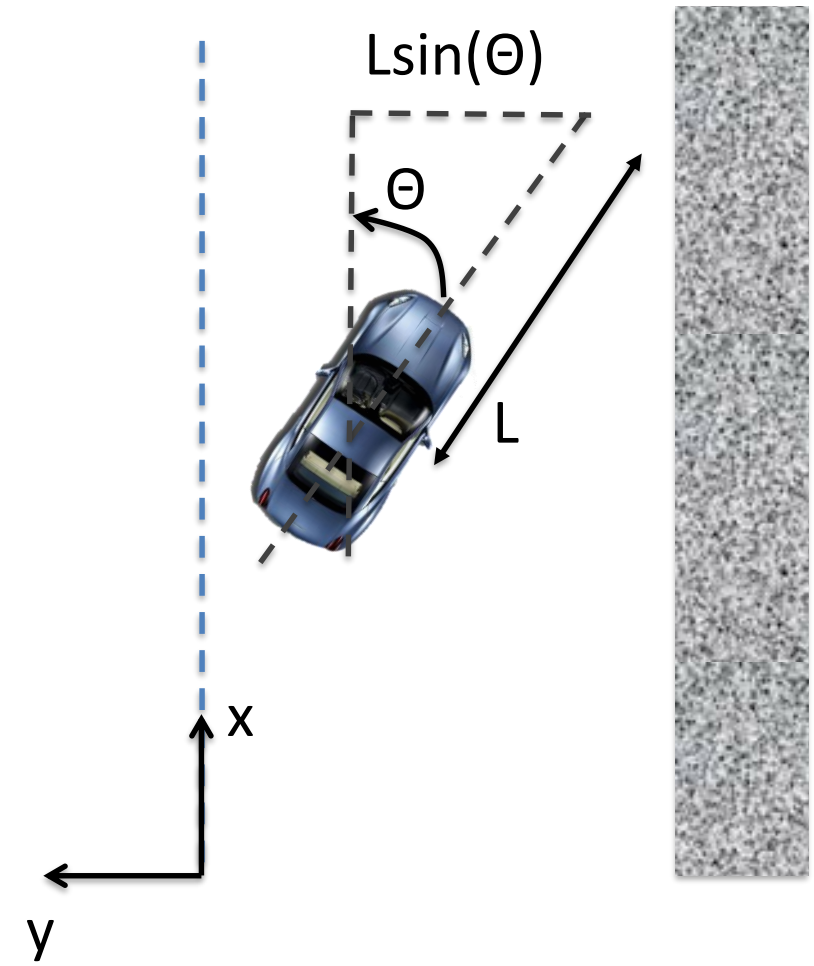
1) keep the car driving along the centerline,

$$y = 0$$

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

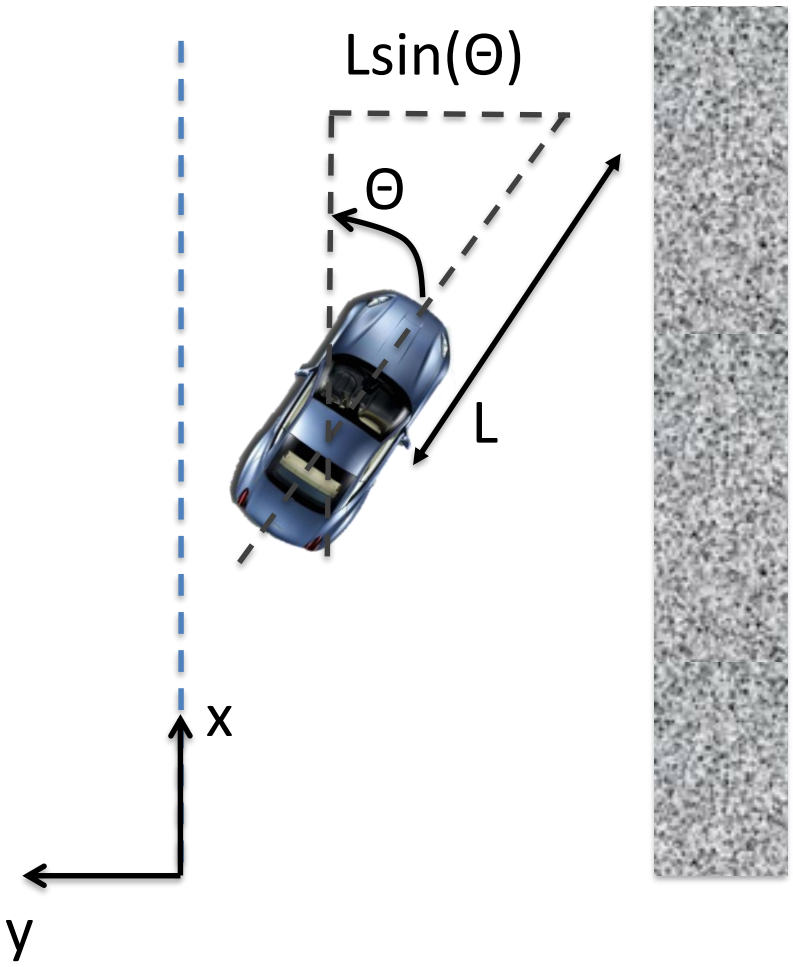
We will hold the velocity constant.

How do we control the steering angle to keep

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

as much as possible?

PID control: error term

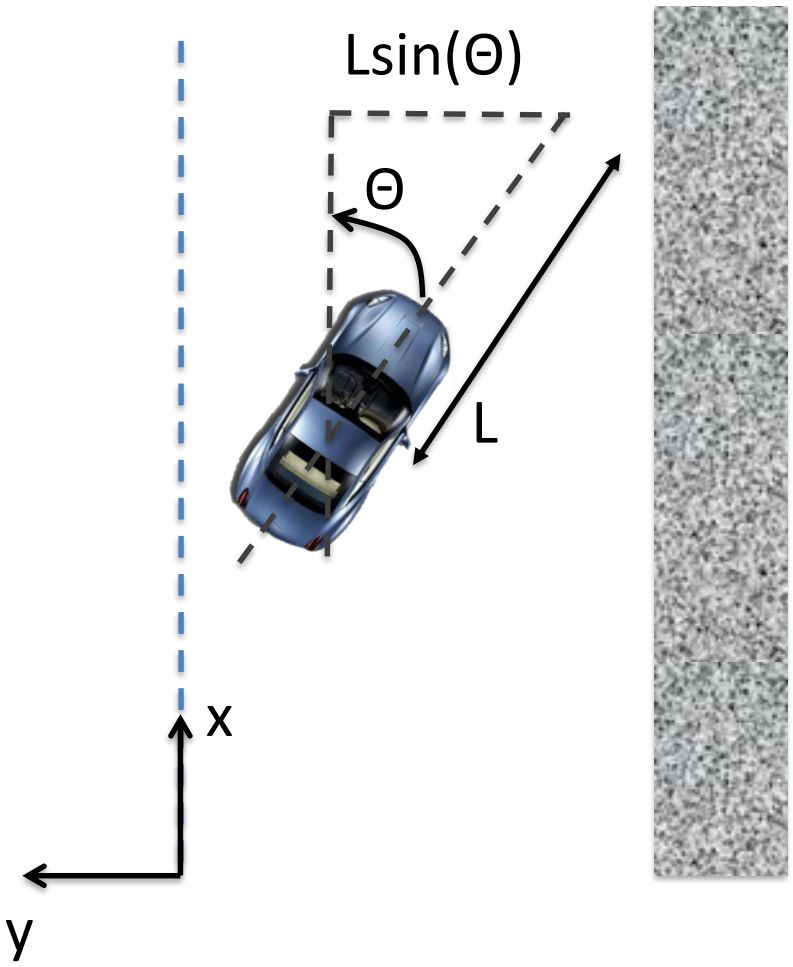


Want both y and Lsin(Θ) to be zero

→ Error term e(t) = -(y + Lsin(Θ))

We'll see why we added a minus sign

PID control: computing input

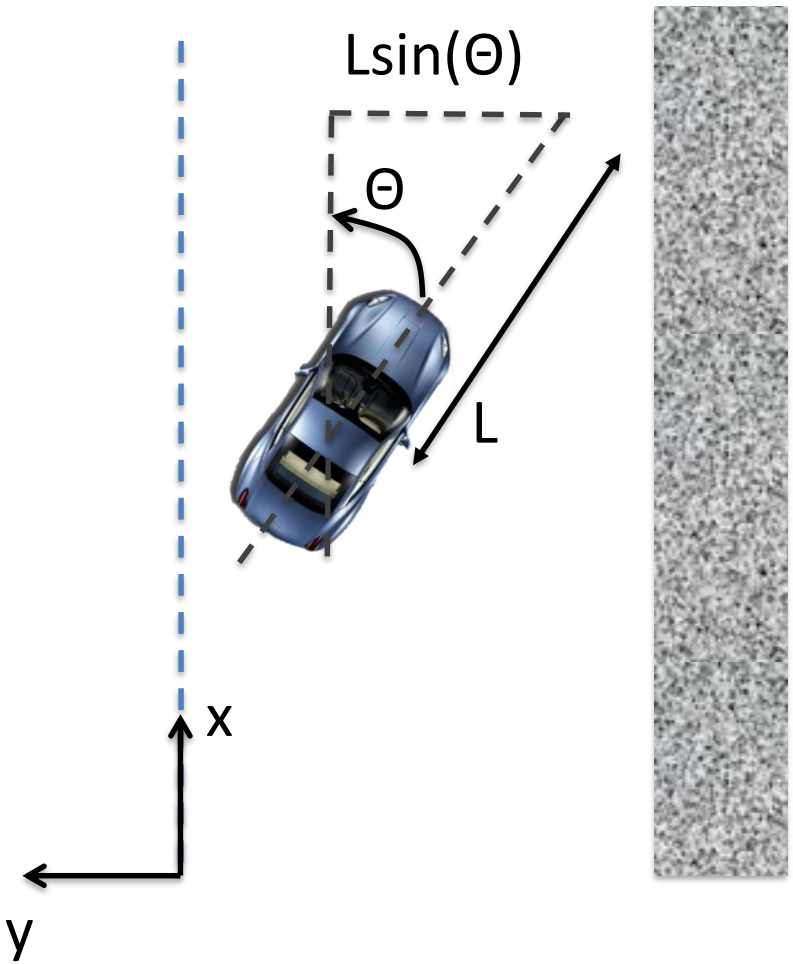


When y > 0, car is to the left of centerline
→ Want to steer right: Θ < 0

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

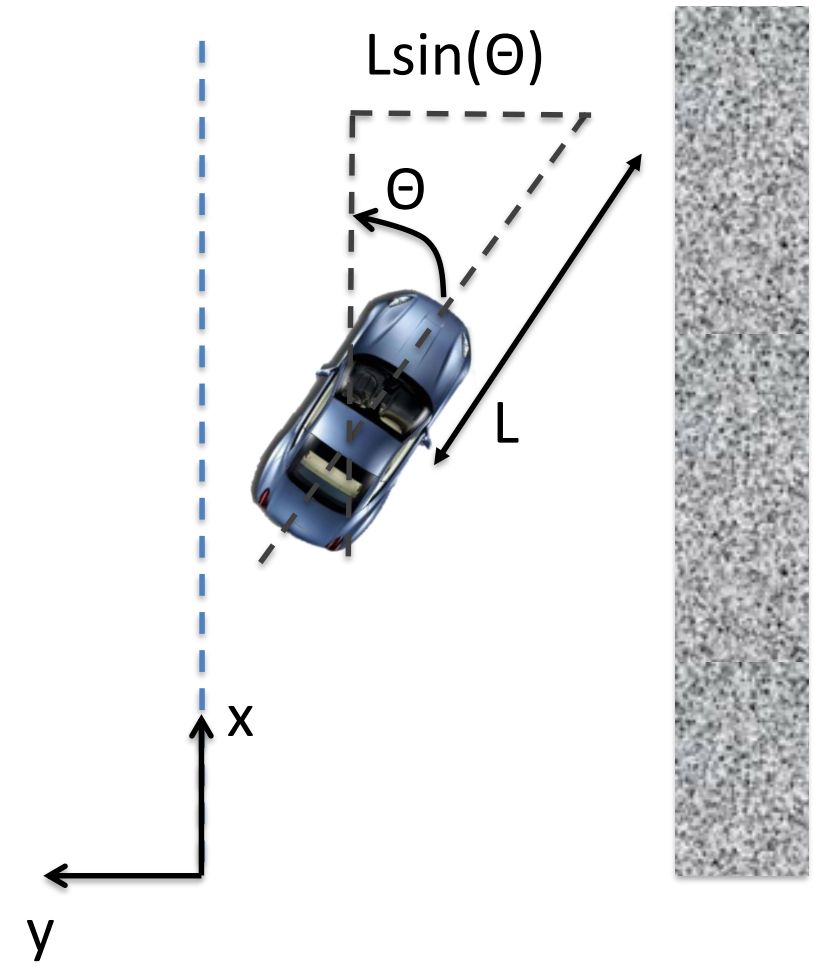


When y < 0, car is to the right of centerline
→ Want to steer left
→ Want Θ > 0

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

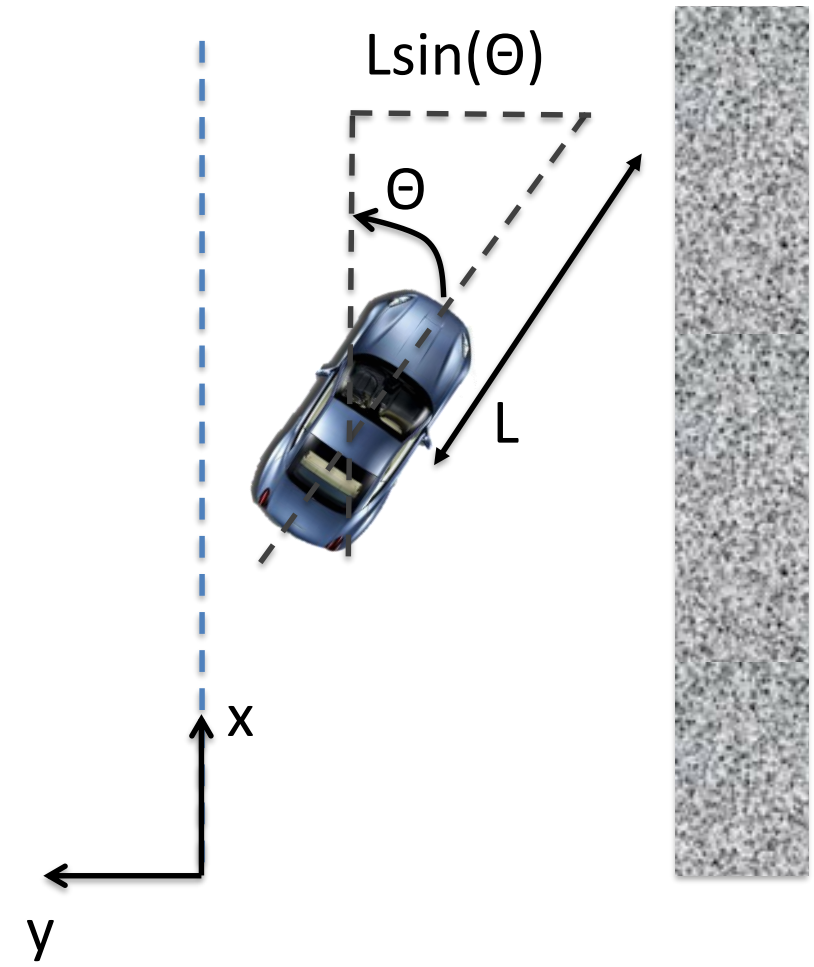


$\Theta_d = C\,K_p\,(-y - L\sin(\Theta)) = C\,K_p\,e(t)$

This is **P**roportional control.

The extra C constant is for scaling distances to angles.

PID control: Derivative control

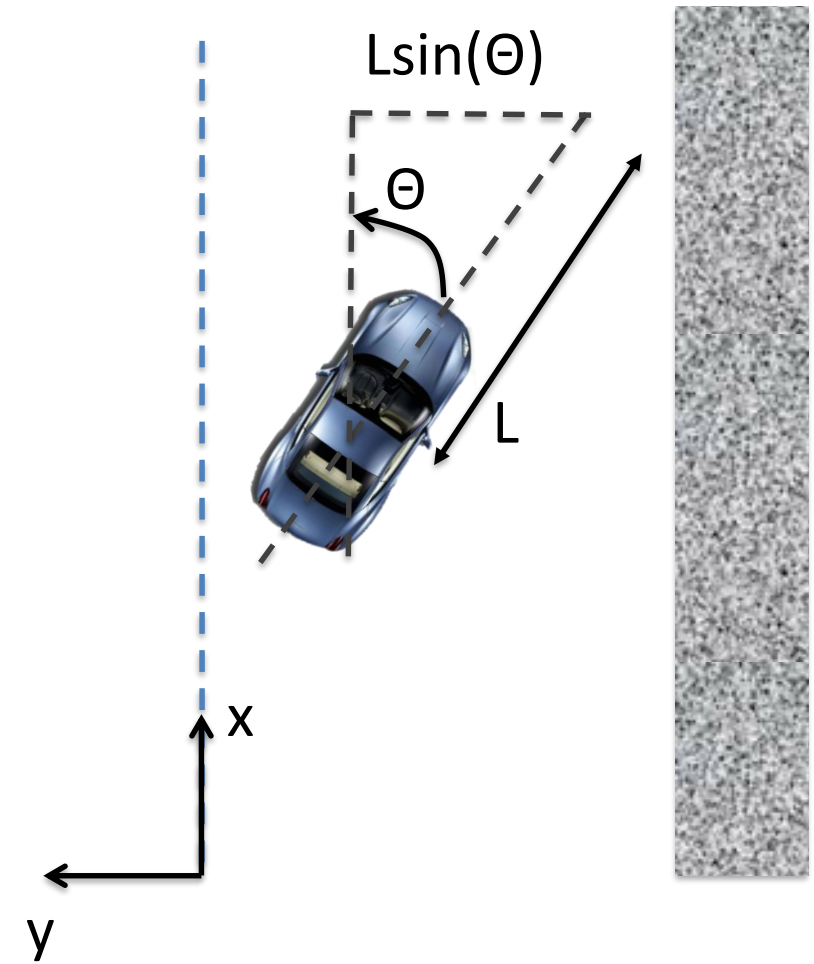


If error term is increasing quickly, we might want the controller to react quickly

→ Apply a *derivative gain*:

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

PID control: Derivative control



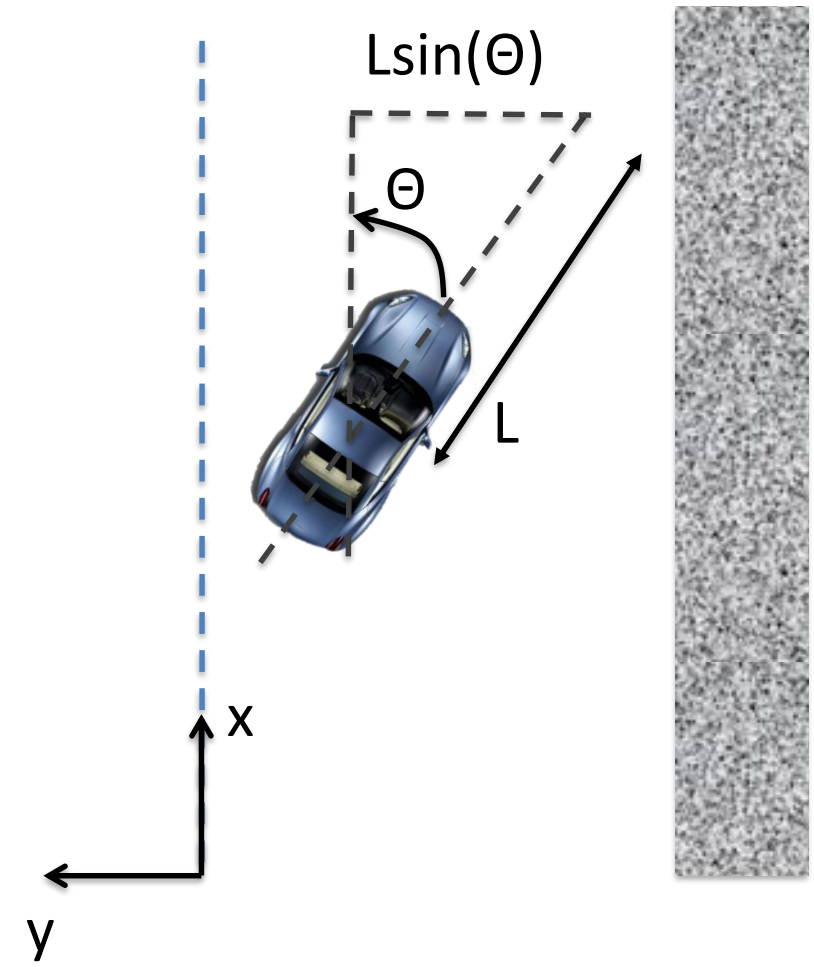
In shown scenario, controller tries to nullify y term.

→ Steer right, $\Theta < 0$ and y decreases → $y' < 0$

→ Error derivative $-y' - L\cos(\Theta) > 0$

→ Trajectory correction takes longer (smoothing out)

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
  - $K_p = 14$
  - $K_i = 2$
  - $K_d = 0.09$

Perception, Planning and Control

- At the end of this week, your car will be driving in a straight line.
- Next week, we see how to extend these algorithms to close a loop on a track.