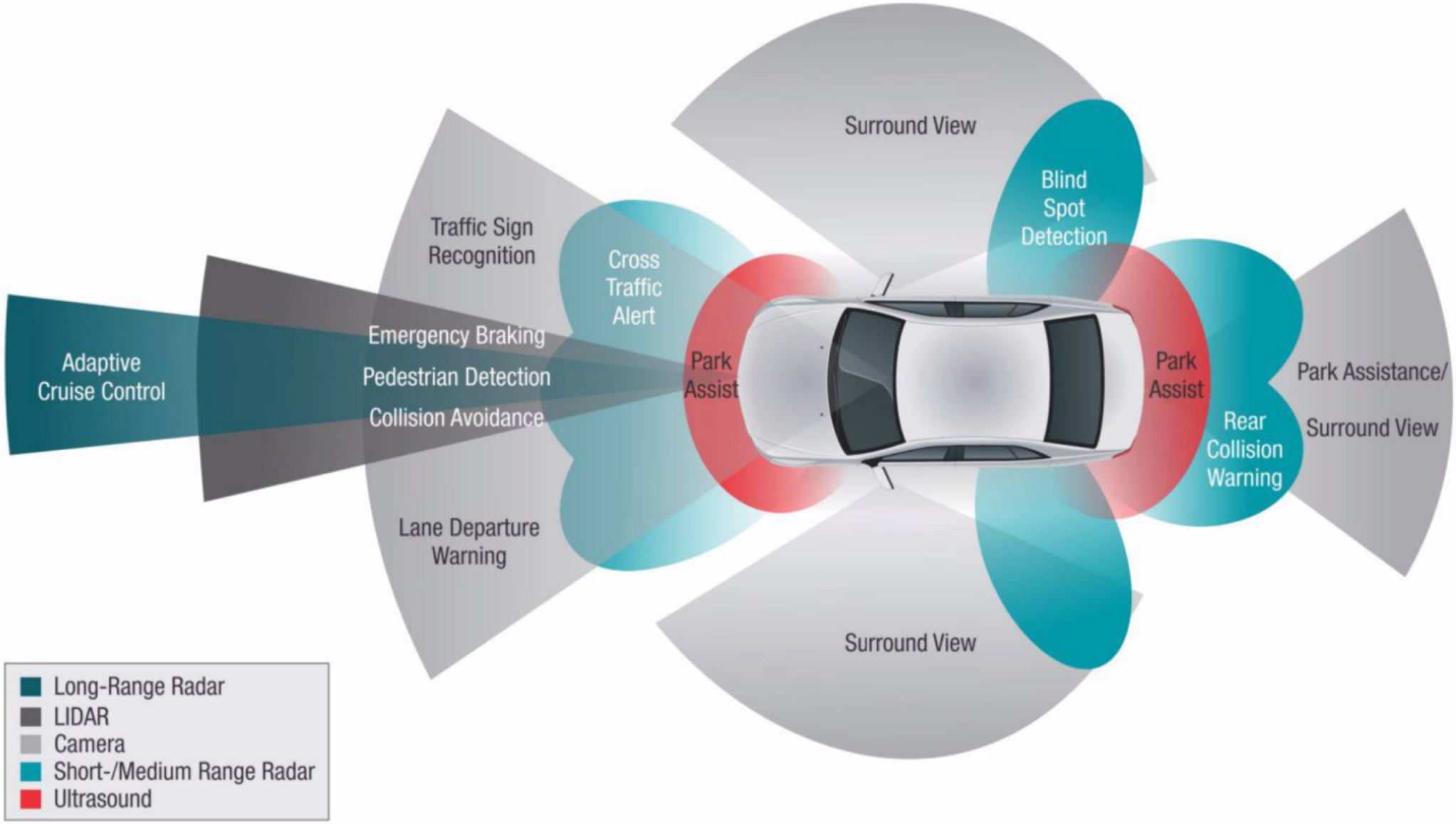
FTC Autonomous

1.0

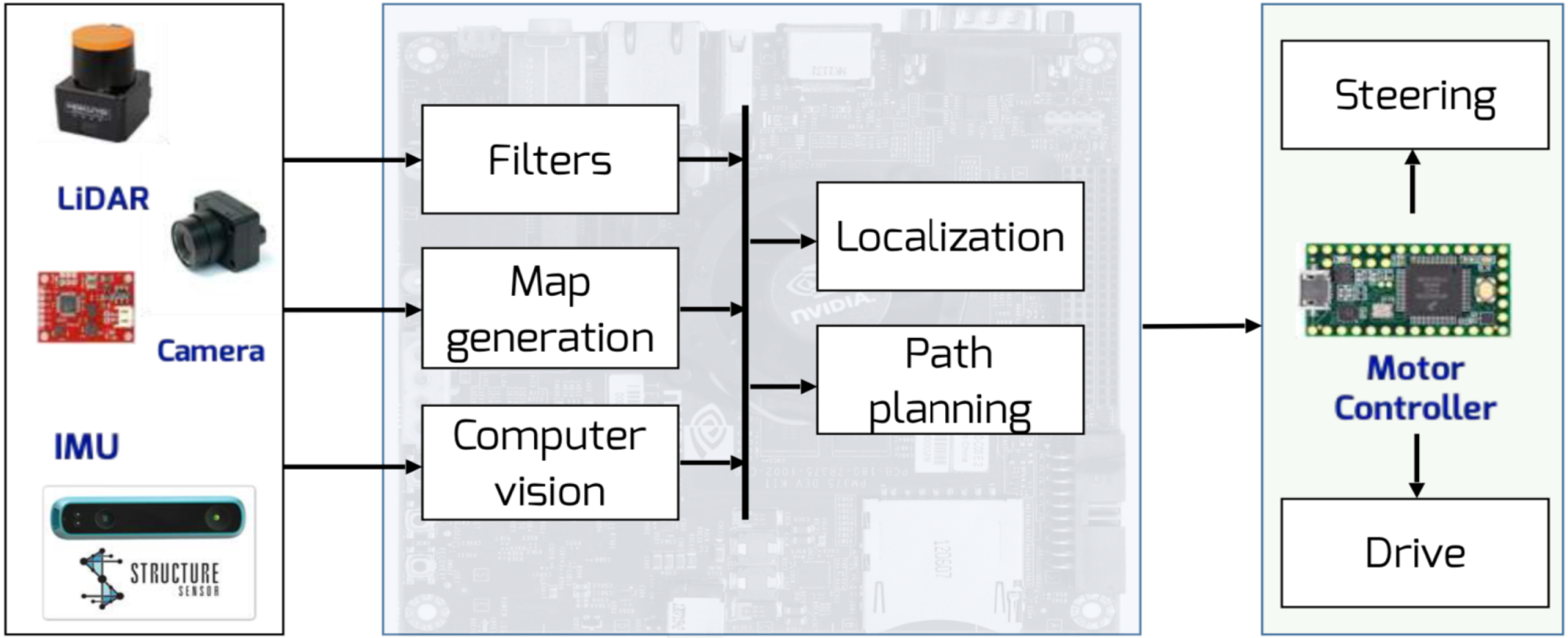
Agenda

- **Autonomous system overview**
- ROS
- Lidar introduction
- PID
- Wall follow implementation
- QA

Autonomous Systems overview



Autonomous Systems overview



Agenda

- Autonomous system overview
- **ROS**
- Lidar introduction
- PID
- Wall follow implementation
- QA

Challenges for software

- **Complexity**
  - Interconnected algorithms
  - Significant internal state
  - Concurrency
  - Multi-rate and asynchronicity
  - Multi-agent / distributed
- **Large-scale**
  - Distributed expertise
- **Resilience**
  - Fault isolation
- **Flexibility**
  - Changing configurations
  - Rapid prototyping
- **Difficult algorithms**
  - Real-time
  - Computation, bandwidth, & memory
- **Re-use**
  - Integration into other systems
  - Portability to other robot platforms

Why need ROS




ros.org

Plumbing

- Process management
- Inter-process communication
- Device drivers

Tools

- Simulation
- Visualization
- Graphical user interface
- Data logging

Capabilities

- Control
- Planning
- Perception
- Mapping
- Manipulation

Ecosystem

- Package organization
- Software distribution
- Documentation
- Tutorials

https://www.youtube.com/watch?v=dby--vF0li8

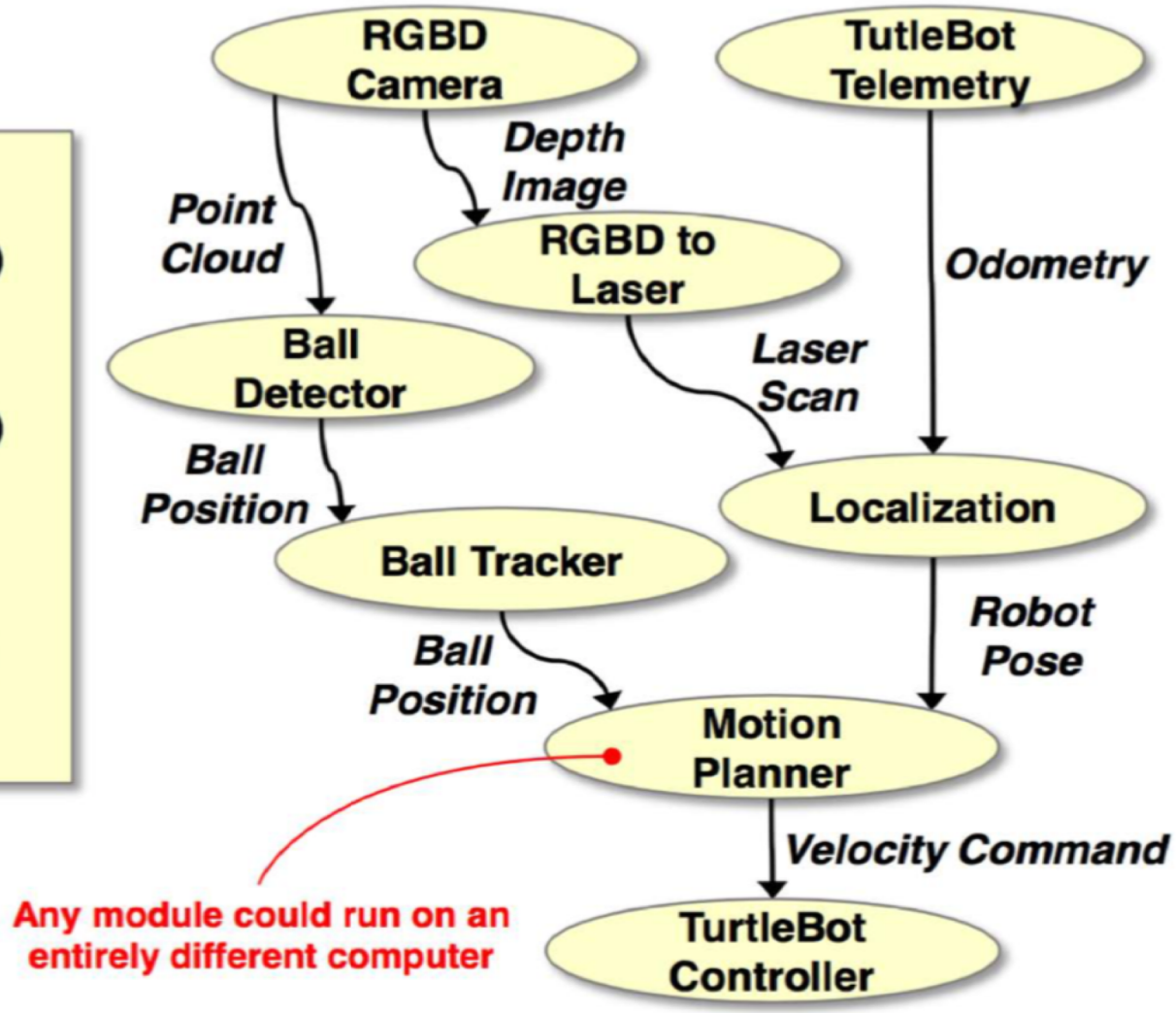
https://www.youtube.com/watch?v=npQMzH3j_d8

ROS advantages

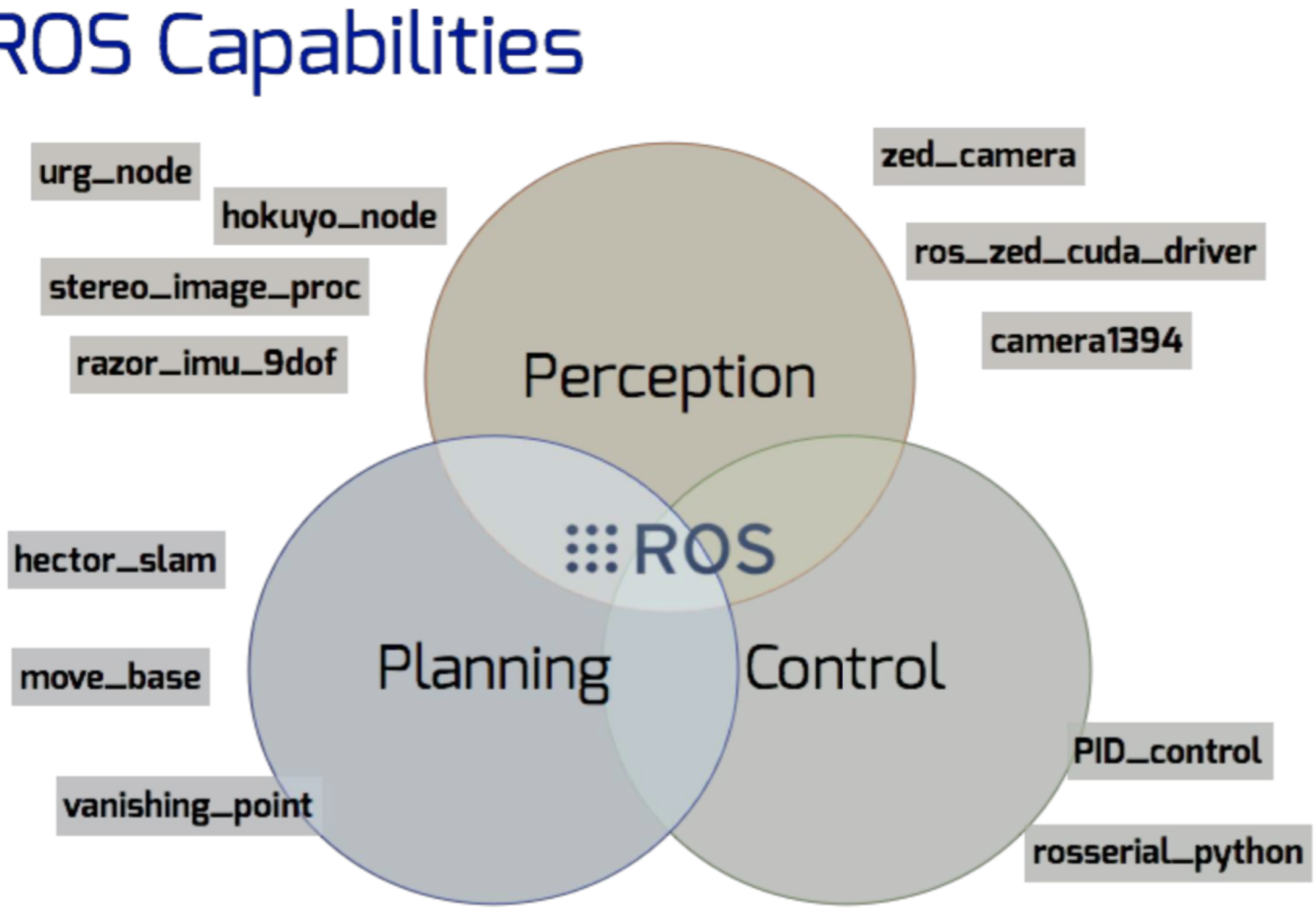
Monolithic, Sequential

```
read_from_rgbd_camera(...)
detect_soccer_ball(...)
update_ball_tracker(...)
convert_rgbd_to_laser(...)
read_odometry(...)
update_localization(...)
update_motion_plan(...)
send_motion_commands(...)
```

Modular, Parallel

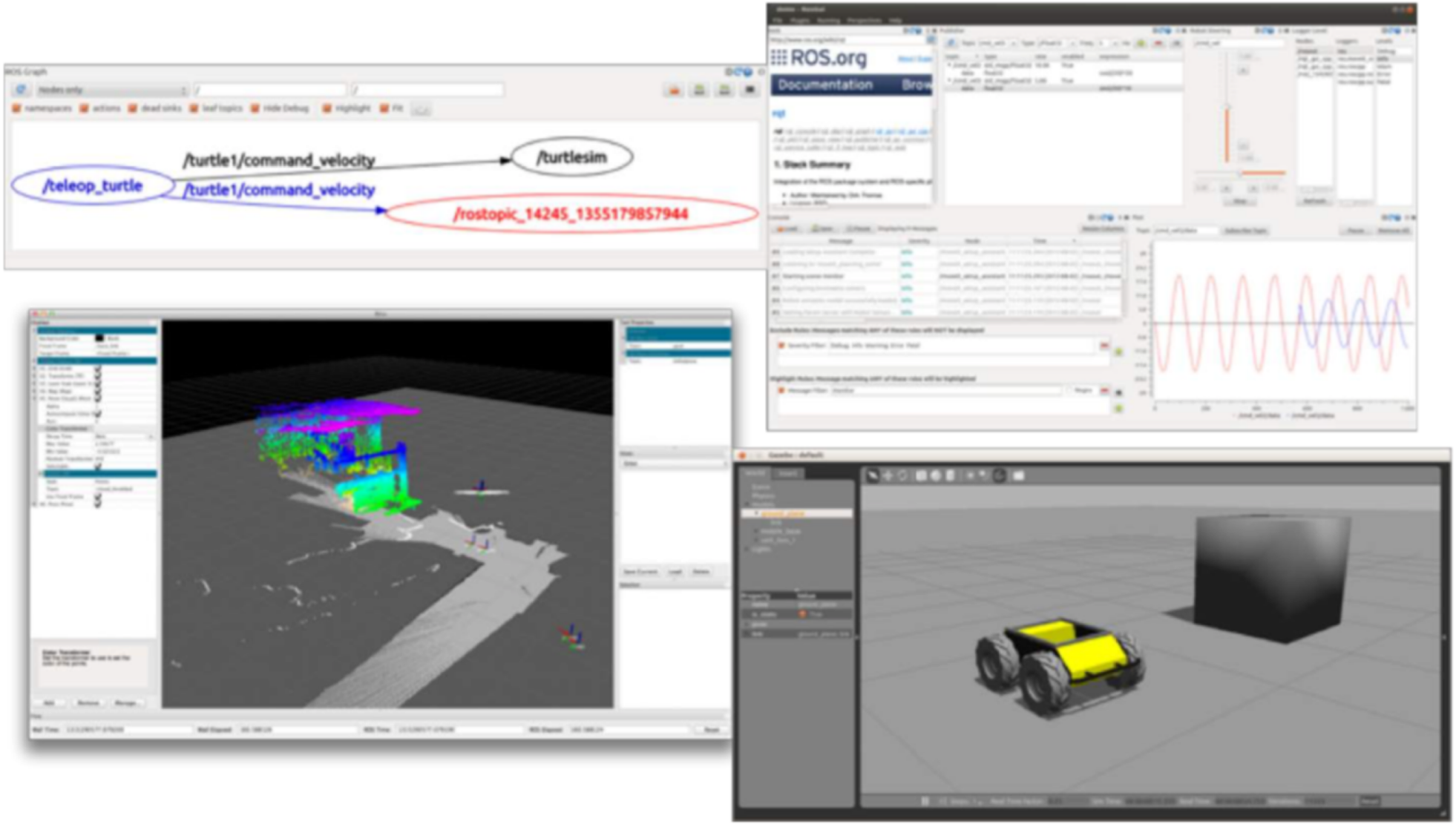


ROS Capabilities



ROS Capabilities

Visualization, debugging and diagnostics, logging, and simulation

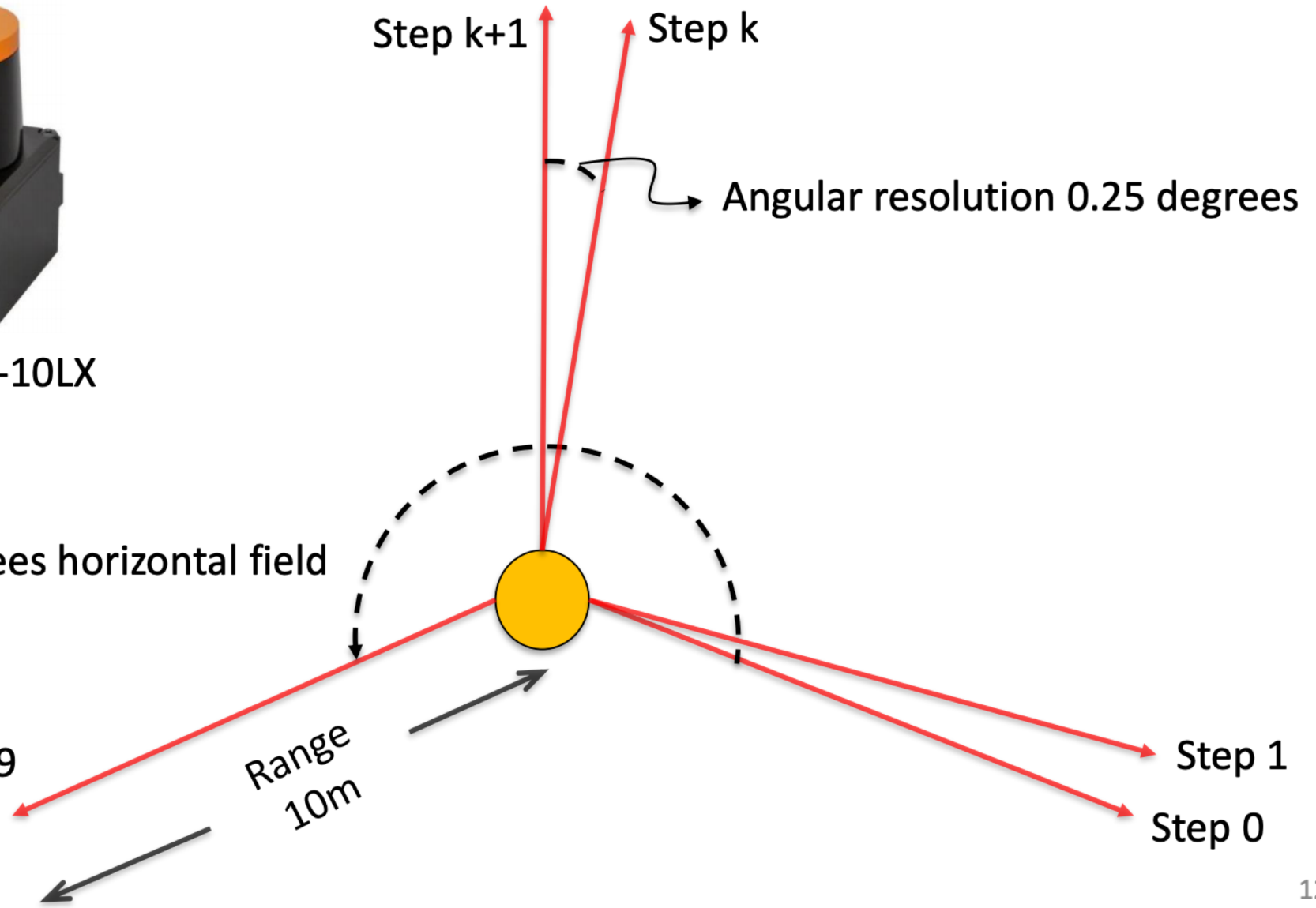


Agenda

- Autonomous system overview
- ROS
- **Lidar introduction**
- PID
- Wall follow implementation
- QA

Integral control

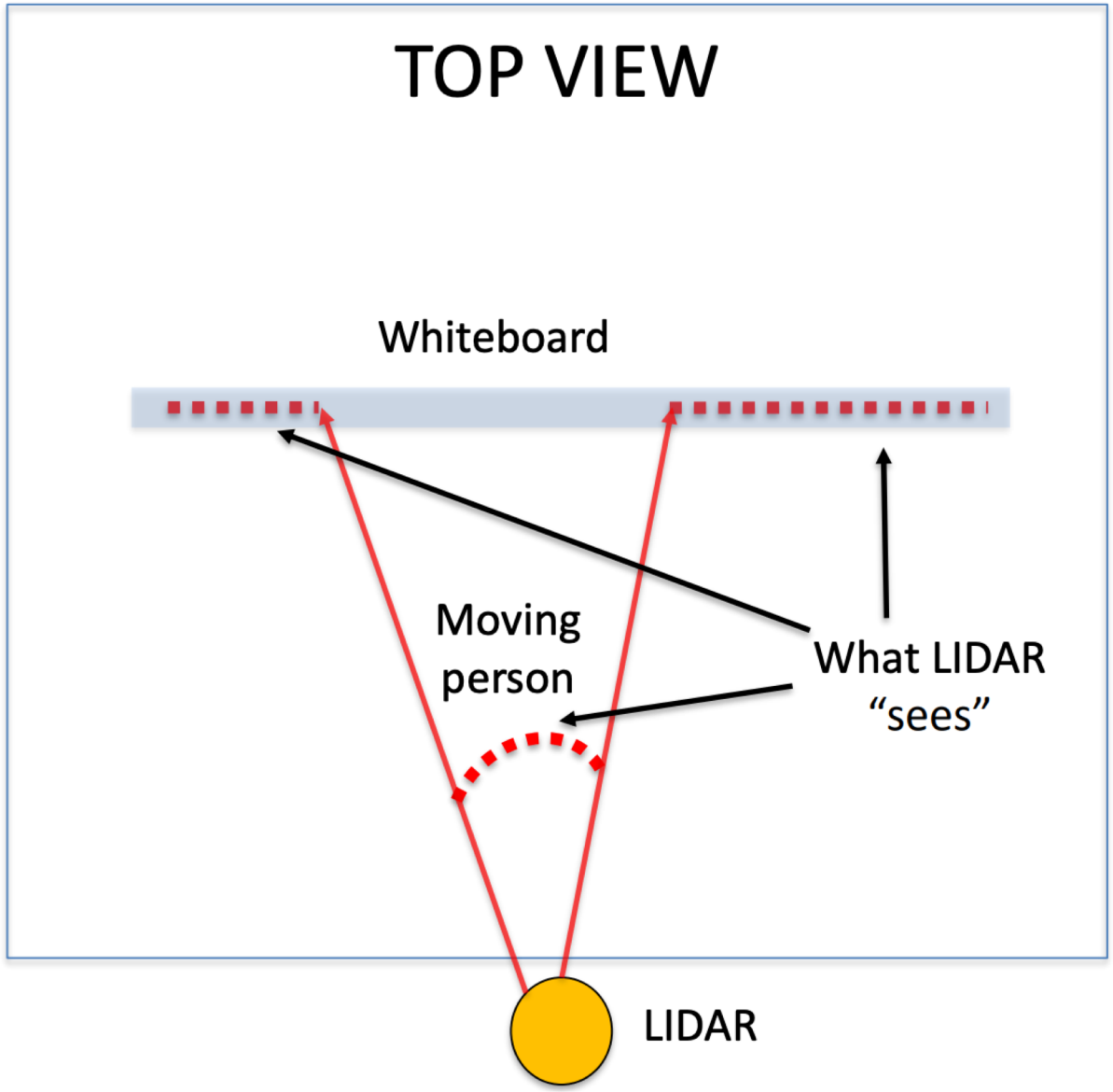
Hokuyo UST-10LX



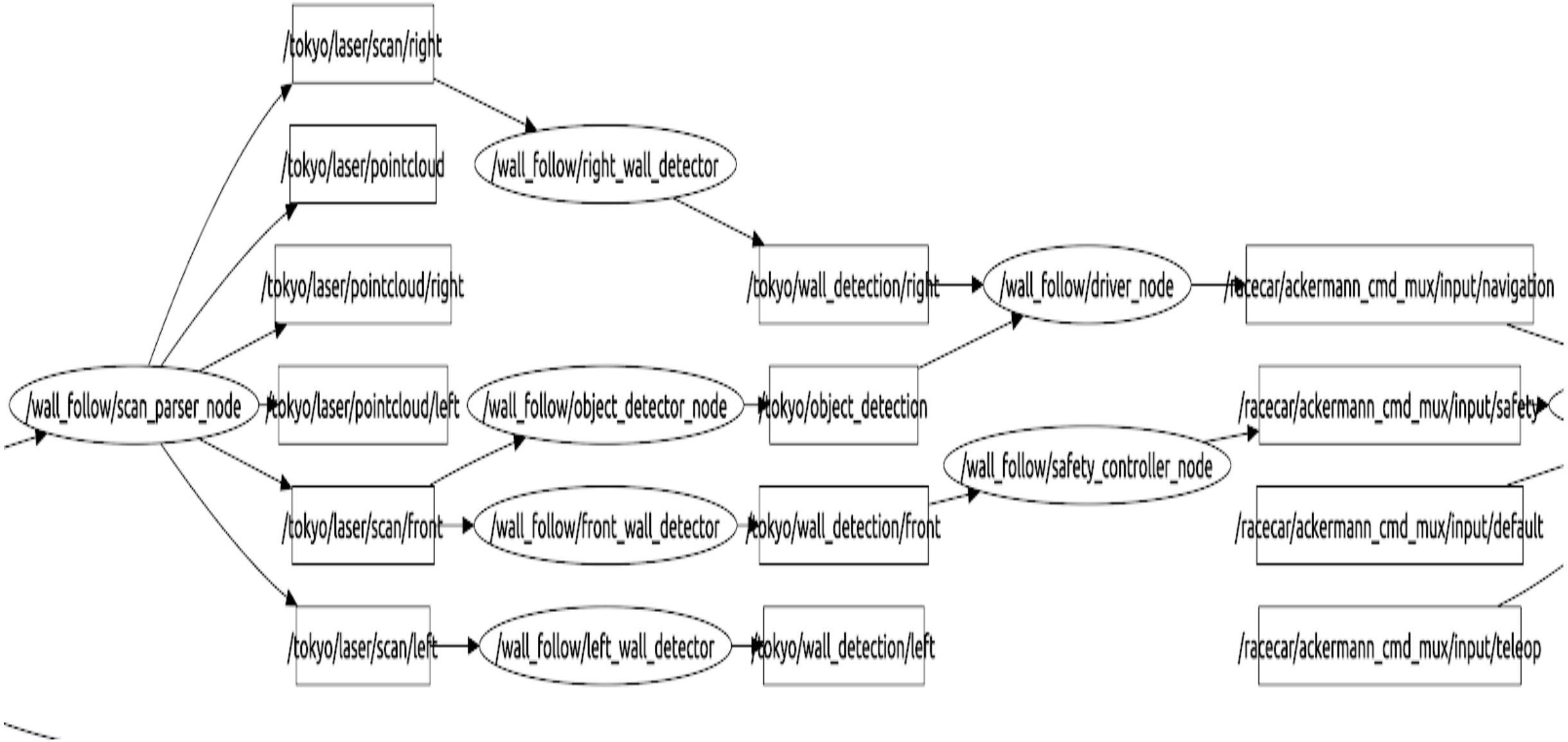
Lidar simple scan sequence

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

```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```



Integral control



Agenda

- Autonomous system overview
- ROS
- Lidar introduction
- **PID**
- Wall follow implementation
- QA

PID Steering control

$$V_\theta = K_p \times e(t) + K_i \times \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

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

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

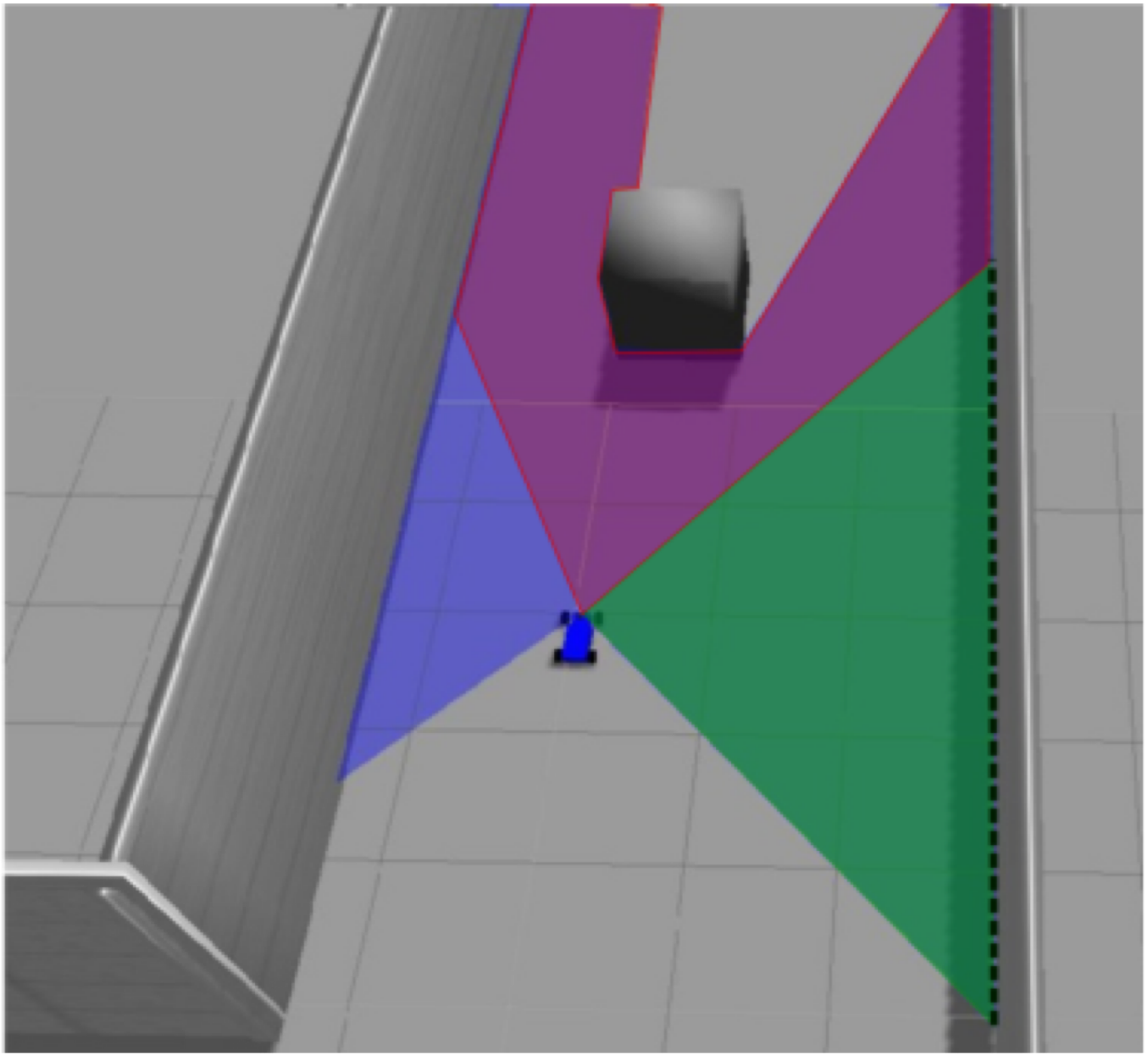
$$steering\ angle = steering\ angle - V_\theta$$

Agenda

- Autonomous system overview
- ROS
- Lidar introduction
- PID
- **Wall follow implementation**
- QA

Wall follower

1. Obstacle Detection
2. Wall Detection
3. Safety Controller
4. Steering Controller

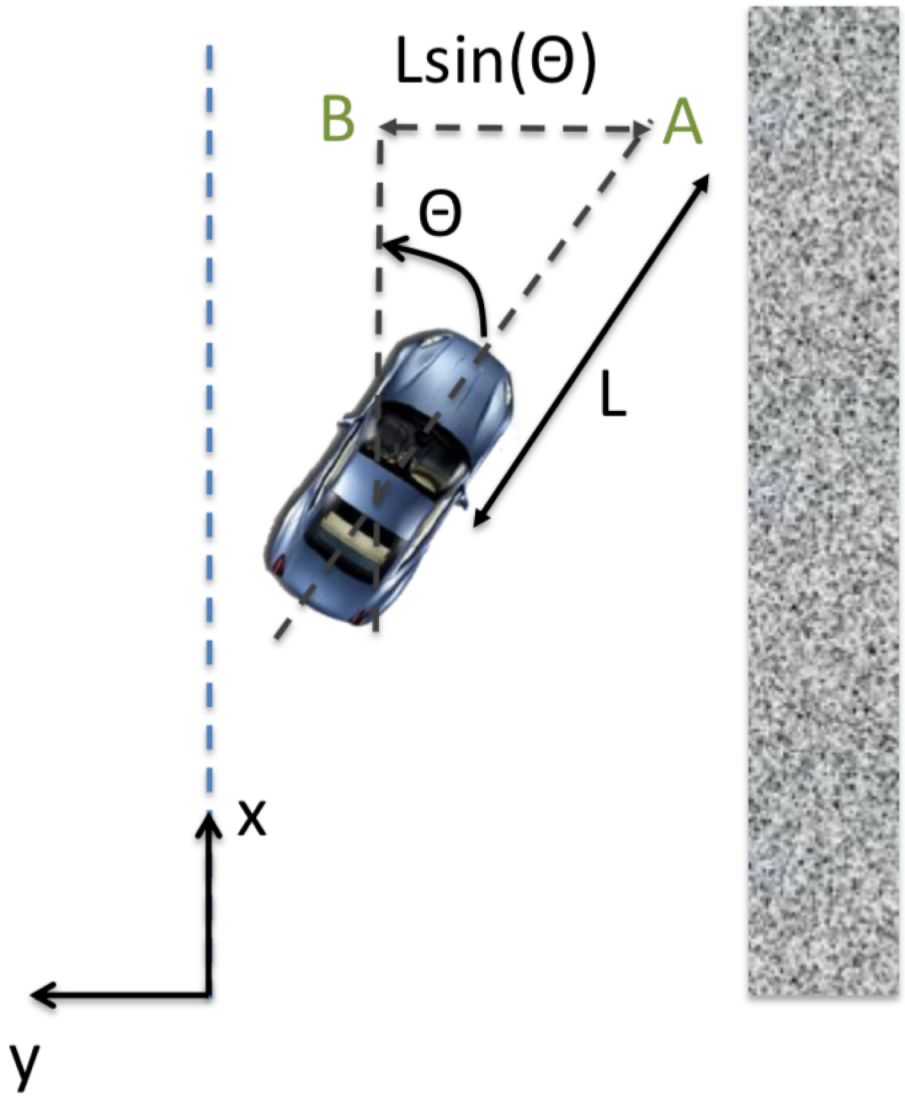


Lidar messages

The wall detector detects the nearest point to the obstacle in trimmed scanner data from the right side, publishing the location of this point in cartesian space relative to the frame of the robot. The desired quantities of angle and range to the wall are approximated by the angle and range of this shortest beam, recoverable from the cartesian coordinates simply by converting back to polar

The wall follower node implements a simple two-state proportional controller to follow a wall to its right at a fixed distance. Given the input vector from the wall detector, it first extracts the distance and angle to the wall encoded in the point message. Given some target distance and angle from the wall, the node multiplies each error by a gain constant and publishes the result to the racecar’ s drive system as the angle component of the ackermann command message.

Control objectives



Control objective:

1) keep the car driving along the centerline,

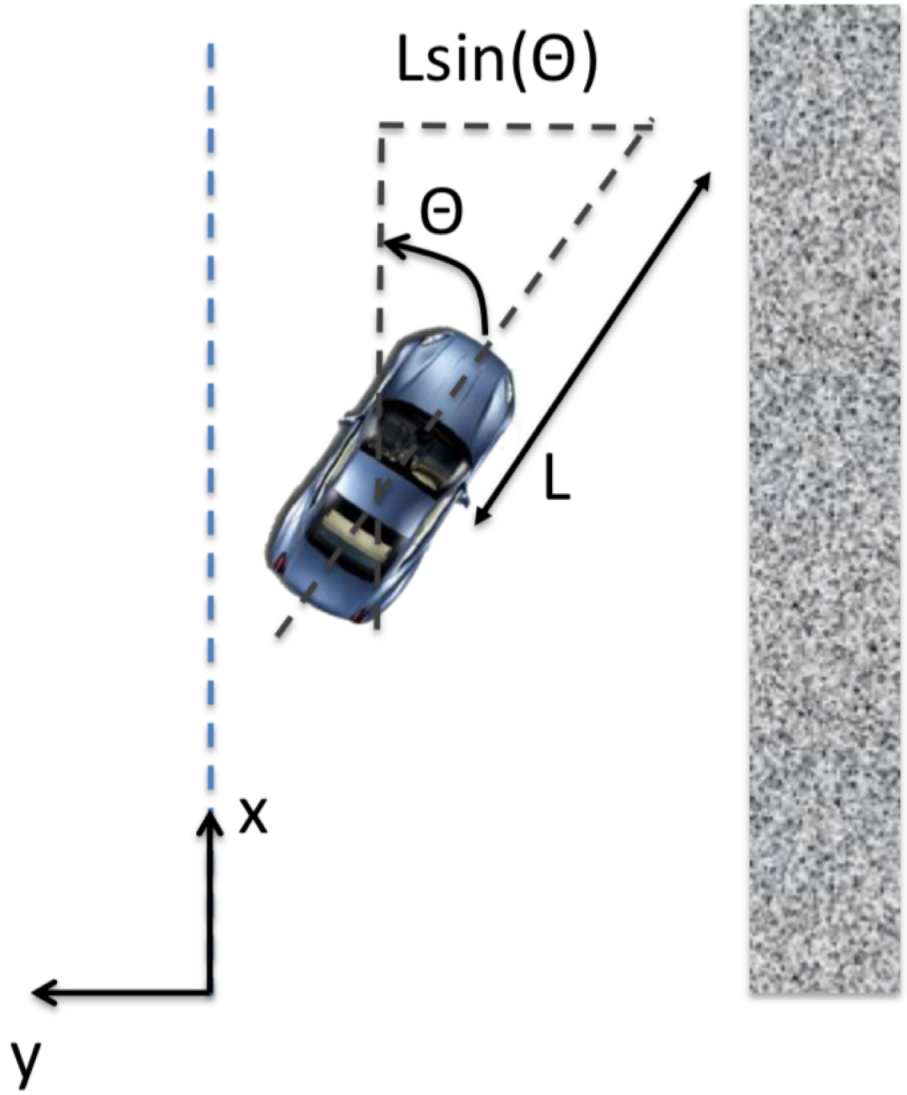
$$y = 0$$

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

Horizontal distance after driving L meters

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

Control objectives



Control input:
Steering angle Θ

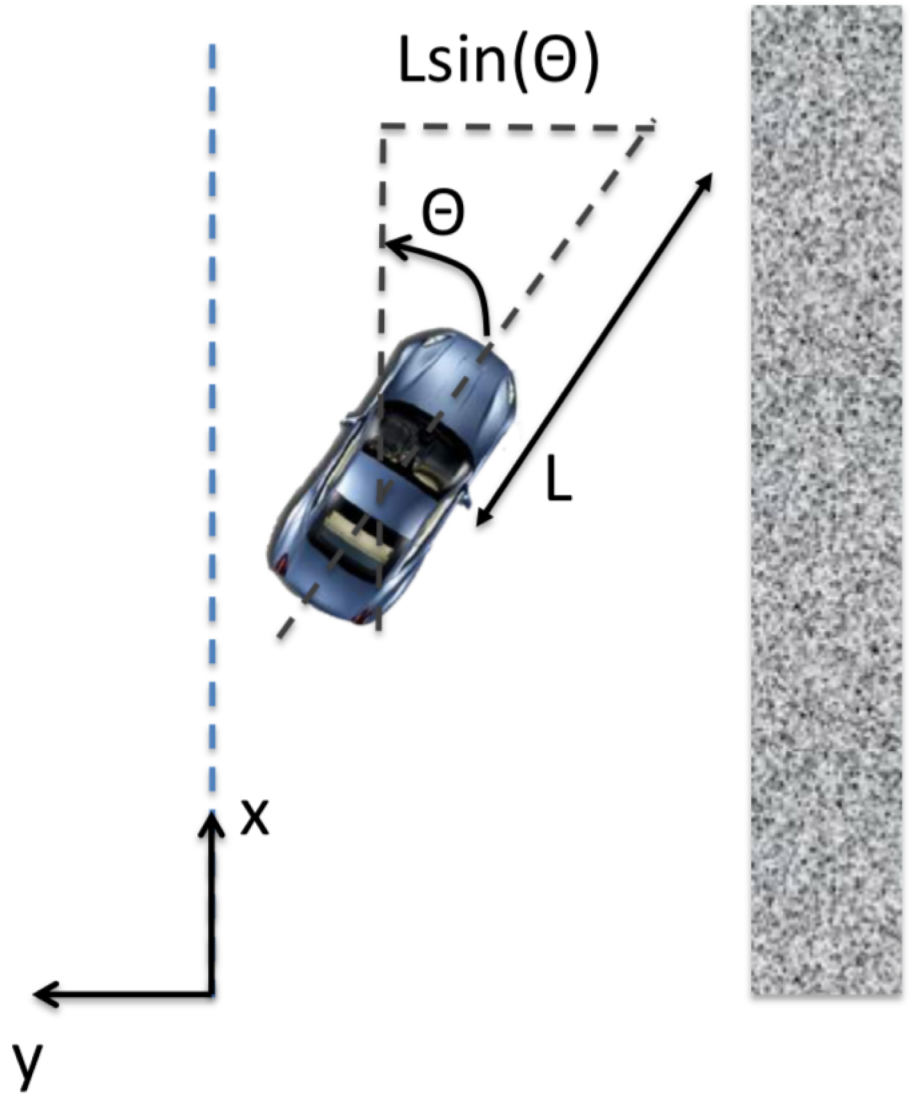
We will hold the velocity constant.

How do we control the steering angle to keep

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

as much as possible?

Control objectives

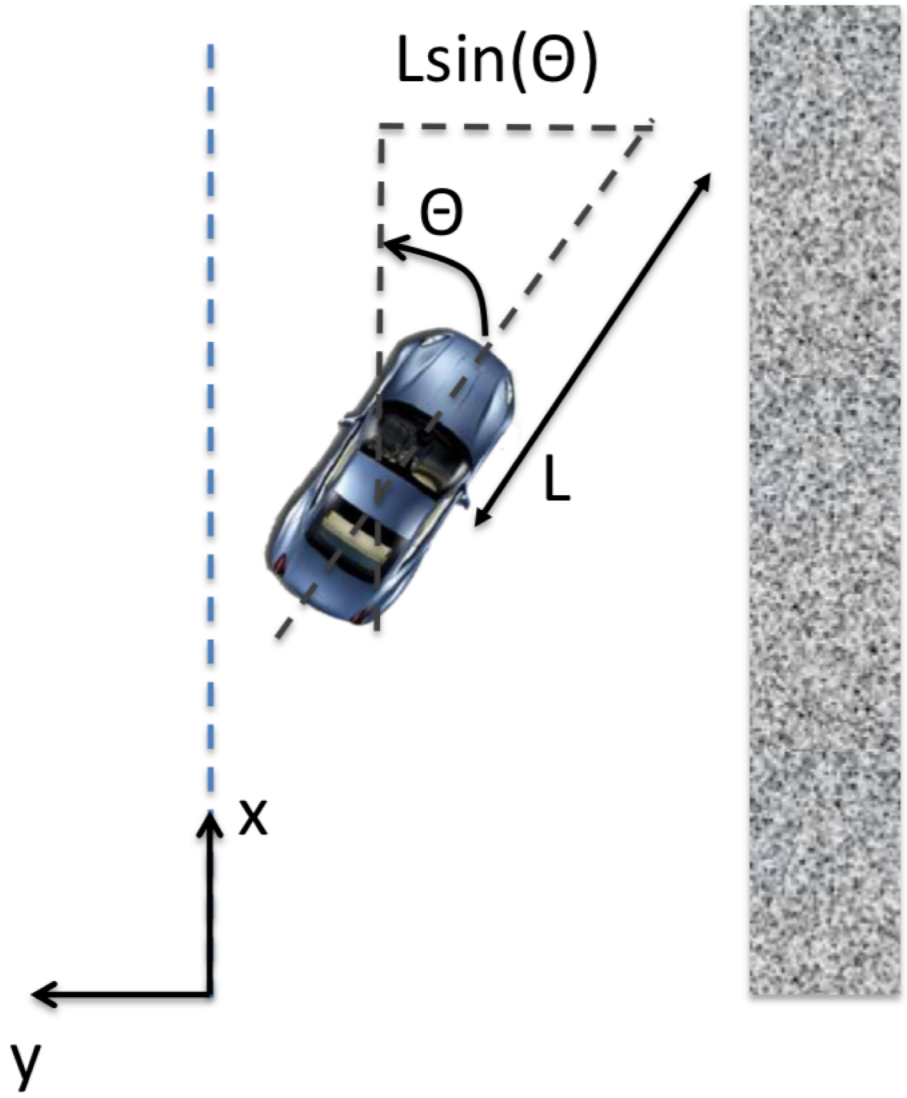


Want both y and Lsin(Θ) to be zero

→ Error term e(t) = -(y + Lsin(Θ))

We'll see why we added a minus sign

Control objectives



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

→ Want to steer right: Θ < 0

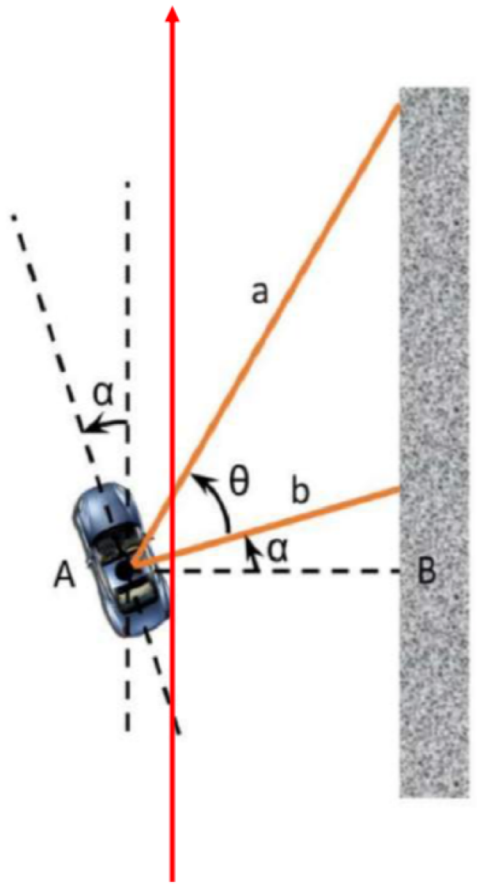
When Lsin(Θ) > 0, we will be to the left of centerline in L meters

→ so want to steer right: Θ < 0

Set *desired* angle to be

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

Forward motioin of car

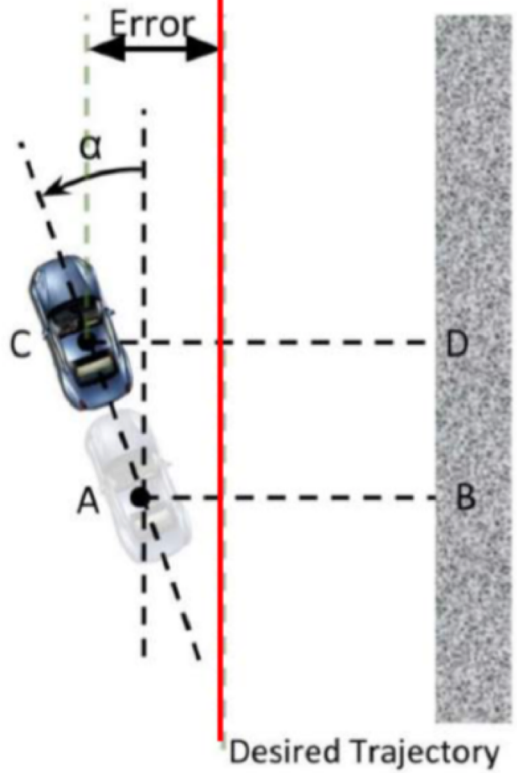


$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta)-b}{a\sin(\theta)}\right)$$

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

Error = desired trajectory – CD

Figure 3: Projecting the car future in time



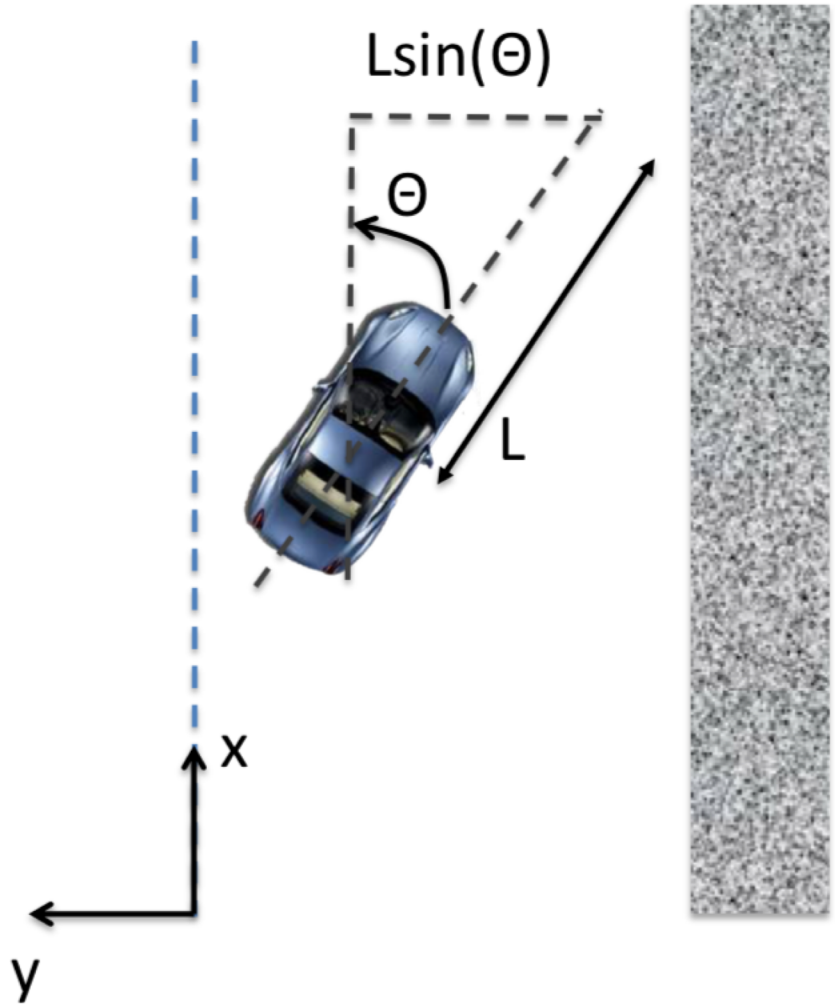
$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta)-b}{a\sin(\theta)}\right)$$

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

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

Propotional control

keep the car driving along the centerline



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

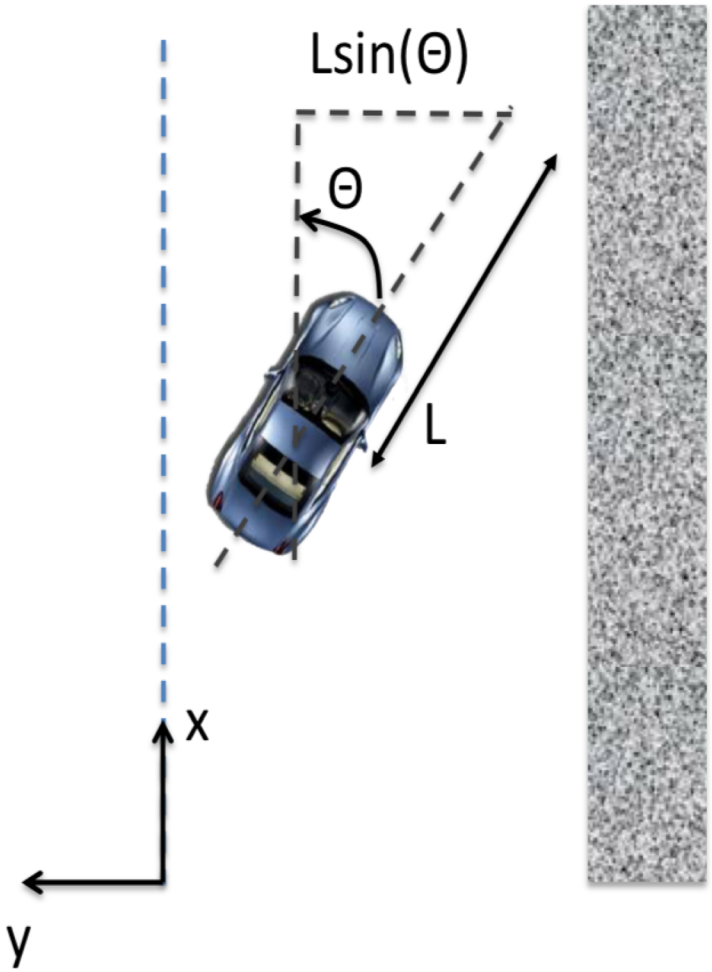
→Want Θ > 0

Consistent with previous requirement:

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

$\Theta_d = CK_p\ (-y - Lsin(\Theta)) = CK_p\ e(t)$

Derivative control

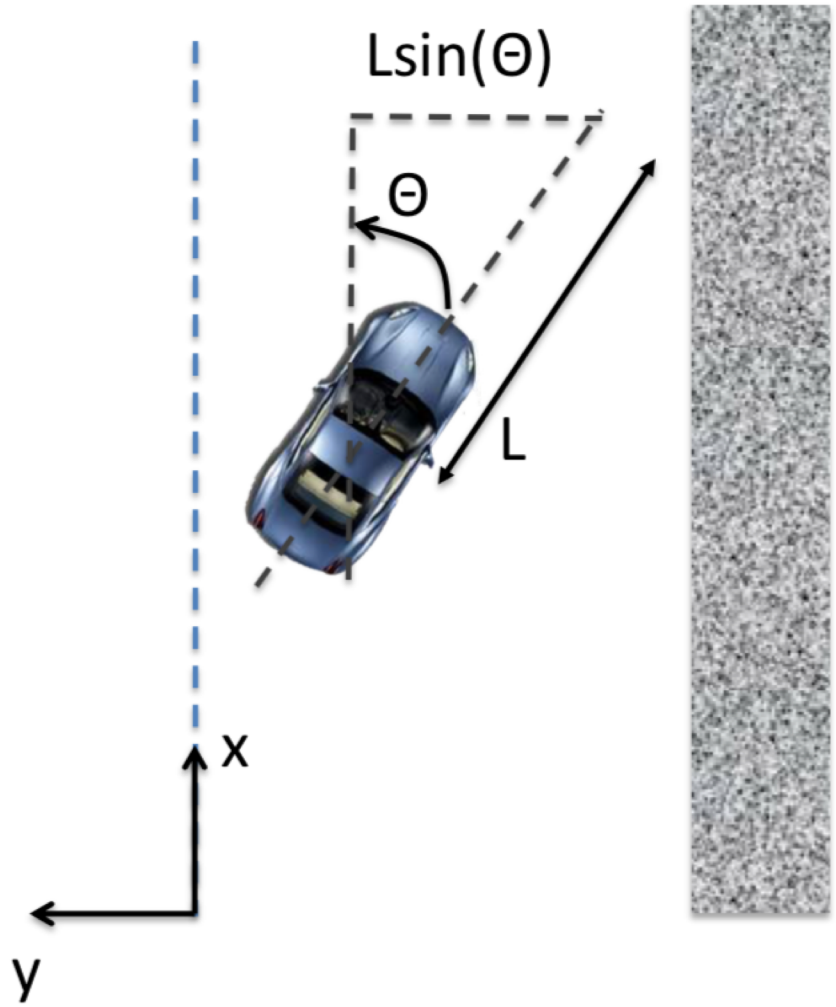


If error term is increasing quickly, we might want the controller to react quickly

→ Apply a *derivative gain*:

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

Integral control

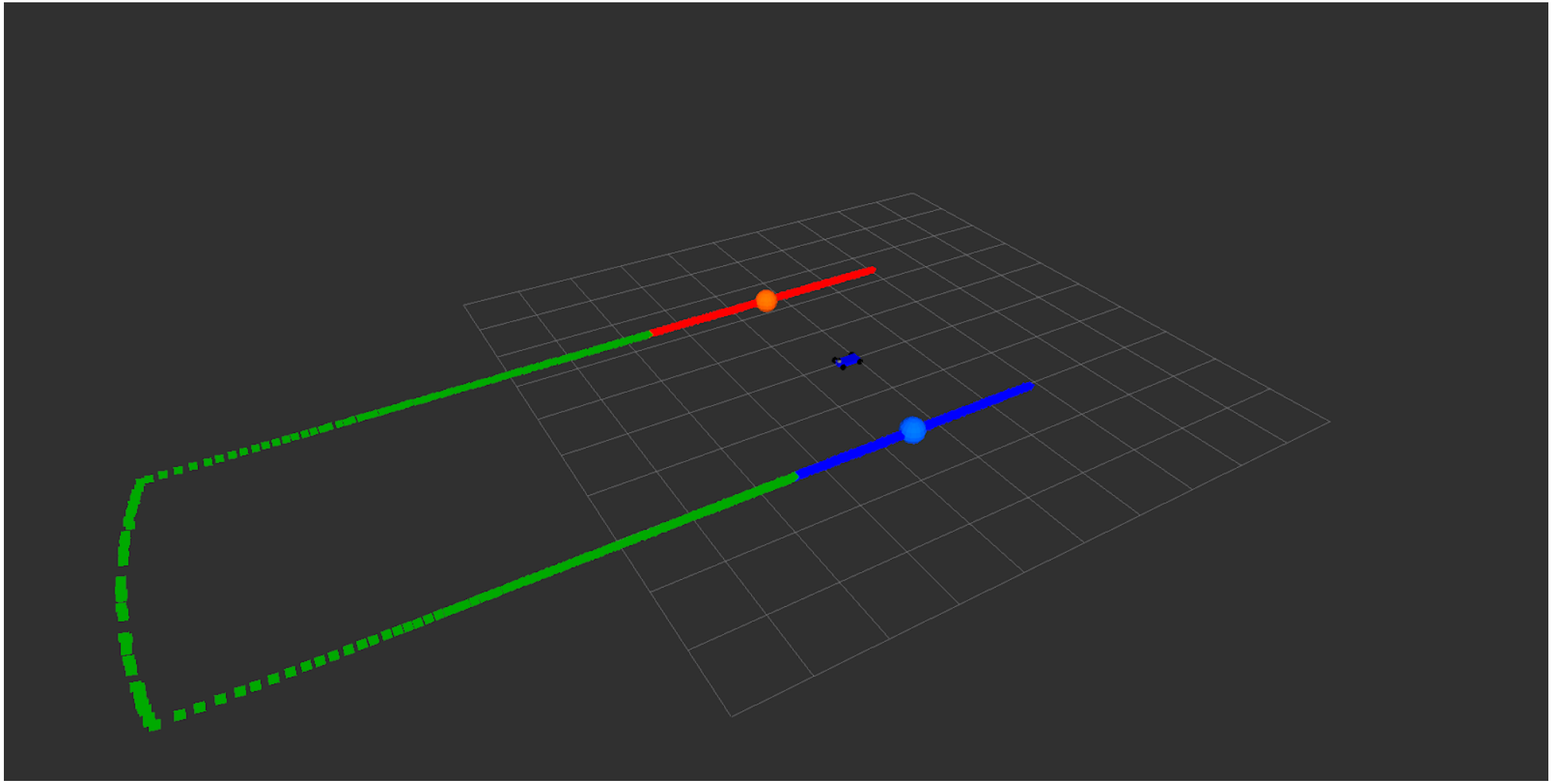


Integral control is proportional to the *cumulative* error

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

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

Estimation Results - Simulator



Integral control

```
    def compute_pd_control(self):
          if self.received_data:
              # given the computed wall slope, compute theta, avoid divide by zero
  error
              if np.abs(self.m) < EPSILON:
                  theta = np.pi / 2.0
                  x_intercept = 0
              else:
                  theta = np.arctan(1.0/self.m)
                  # solve for y=0 in y=mx+c
                  x_intercept = self.c / self.m

              # x axis is perp. to robot but not perpindicular to wall
              # cosine term solves for minimum distance to wall
              wall_dist = np.abs(np.cos(theta)*x_intercept)

              # control proportional to angular error and distance from wall
              distance_term = self.direction_muliplier * KP * (wall_dist -
   TARGET_DISTANCE)
              angle_term = KD * theta
              control = angle_term + distance_term
              # avoid turning too sharply
              self.control = (np.clip(control, -0.3, 0.3), SPEED)
```

QA