Interdisciplinary Centre for
Security, Reliability and Trust

Connected and Automated Driving

Final Project Introduction

François Robinet
francois.robinet@uni.lu

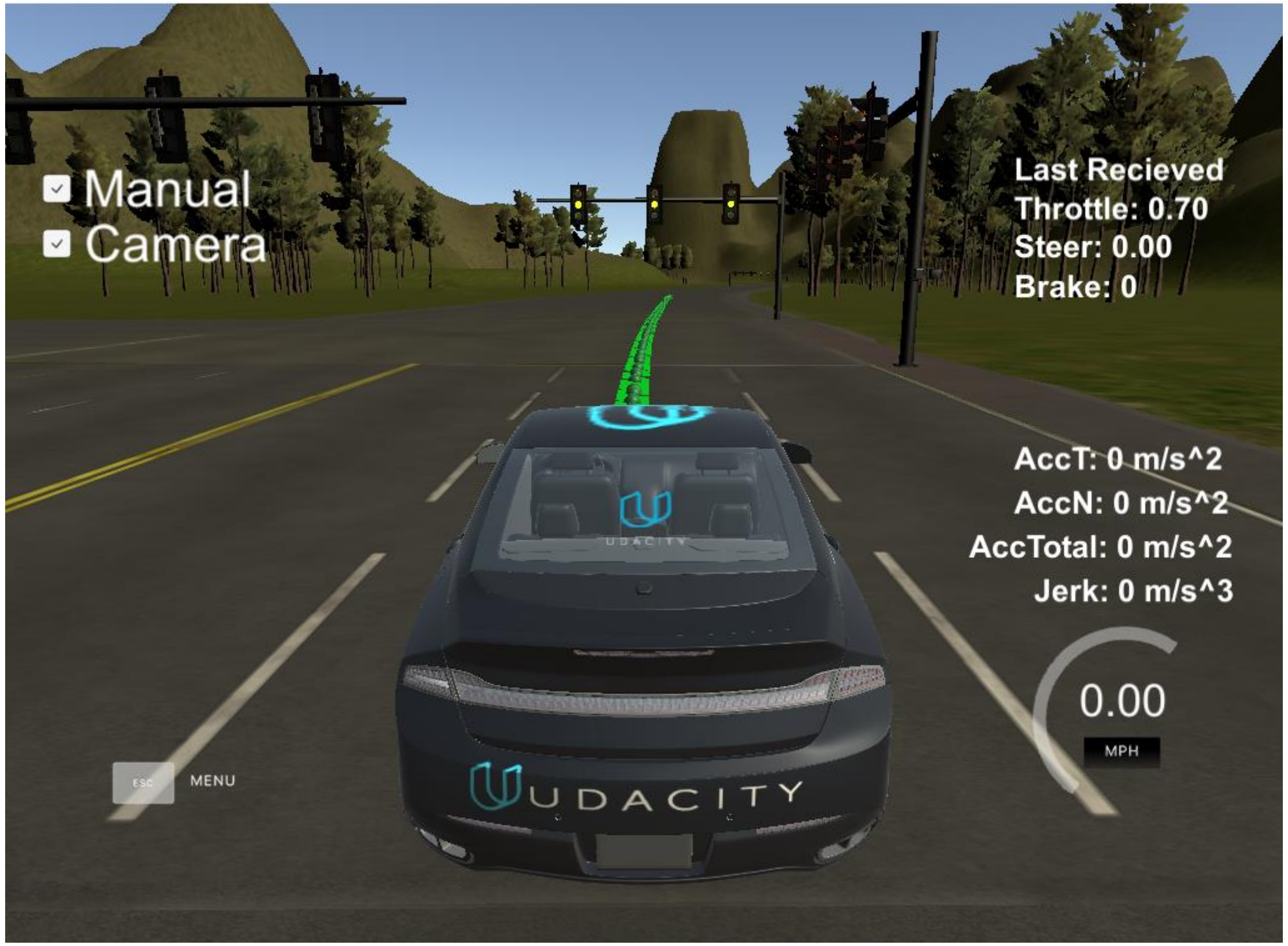
Plan

- Introduction
- Evaluation
- Software Environment
- Project Walkthrough
- Office Hours

Project: Driving in a Simulator
Perception, Planning and Control

Driving in a Simulator

This project will be using Udacity's simulator



Last Recieved
Throttle: 0.70
Steer: 0.00
Brake: 0

AccT: 0 m/s^2
AccN: 0 m/s^2
AccTotal: 0 m/s^2
Jerk: 0 m/s^3

0.00

Driving in a Simulator

Assumption: perfect localization of the car inside a known map

The following is known:

- Pose of the car
- Shape of the track (ie. Waypoints with target linear speed at each waypoint)
- Position of traffic lights on the track

Tasks

- Perception: detect traffic lights and their state
- Planning: find a path that follows the track waypoints while smoothly braking stopping on red lights
- Control: implement controllers for both steering and accelerating (including braking)

Software Environment

Software Environment

The Robot Operating System (ROS)

- Not an actual operating system like Linux or Windows, more a software suite for managing processes)
- A process managed by ROS is called a **node**
- ROS Nodes communicate by reading/publishing **messages** from/to t**opics**
    - A node subscribes to a topic by registering a callback method to be called whenever a new message gets sent on the topic
    - This publish/subscribe model allows to design/test/run nodes completely independently from each other (eg. mock their inputs and examine their output)
    - If you know reactive programming or the actor model for concurrent communication, you will feel right at home
- You won't need to know much about ROS to start working on the project as we provide scripts for many things, but getting familiar with it will help you debug efficiently

Software Environment

- We have prepared a custom Docker image to ensure that everyone is running the exact same software
- A Docker image can be seen as a "Virtual Machine Template"
  - You can create a Docker container by "running an image"
  - A container can be seen as a lightweight VM (on Windows and Mac, Docker runs over a tiny Linux kernel, on Linux it uses the native kernel)
  - People who run the same image will have the same software environment, in our case Ubuntu 20.04 with ROS Noetic and all Python 3 dependencies installed

Note: Our container is not the same as the outdated one used by Udacity for their own courses, we are only using their simulator and some of their basic ROS nodes

Software Environment Demo

https://github.com/frobinet/cad-final-project

Evaluation

Project Evaluation

- Reminder: this final project accounts for 50% of the total grade of the class (the remaining 50% are the 4 assignments)
- You may work on this project in teams from 1 to 3 students
  - Send me **a single email per group** so I can reassign each team to the right group on Moodle
  - We will be evaluating the code by running `launch.sh`
- Evaluation Details
  - The car drives around the track while ignoring traffic lights (20%)
  - The car respects groundtruth traffic lights and is able to recover when manually moved slightly off track (25%)
  - The car is able to detect traffic light states using Deep Learning (25%)
  - Final Presentation (30%)
- We modified things quite a bit from Udacity's original project
  Cheating without getting caught is likely much harder than doing the work!

Project Presentation

- 20-minute Presentation
  - Explain your solution and design choices with respect to planning and control
  - If you implemented traffic light perception, detail your solution (data, model, training, …)
  - You won't be penalized in presentation grades for failing to implement the traffic light detection
- Your presentation should include a demo video
  - Prepare a hosted video illustrating a full lap (provide a link to the video since Webex can be too laggy)
  - Show the maximal speed your car can reach while still respecting traffic lights
  - Show whether your car is able to recover from being manually moved slightly off track
- Every member of the group is expected to take part in the presentation
- 10-minute at the end for questions related to the project and the relevant material

Project Presentation

Presentation Date

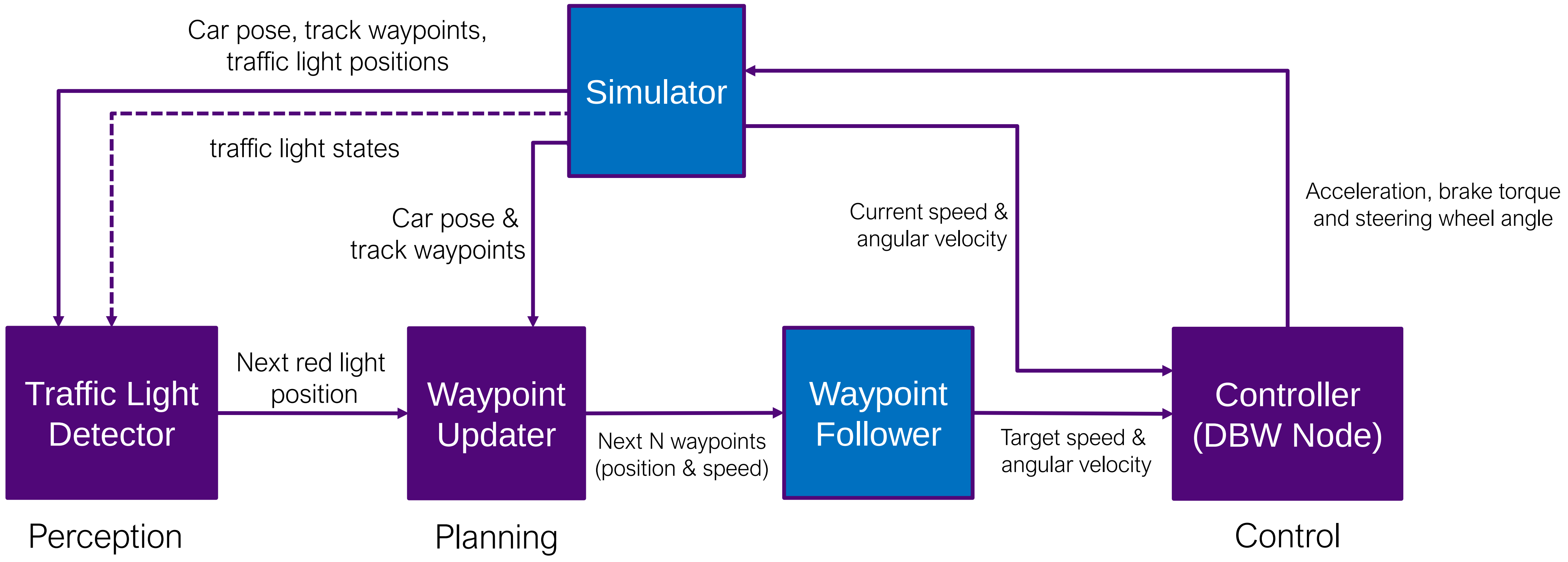
The University has set an official “exam” date on 26th January from 8 to 10am

- Depending on the number of groups, we won’t be able to fit every presentation in this 2-hour slot

- Specific presentation times to be discussed: do you have other exams on Jan 26th?

Project Walkthrough

Overall Architecture

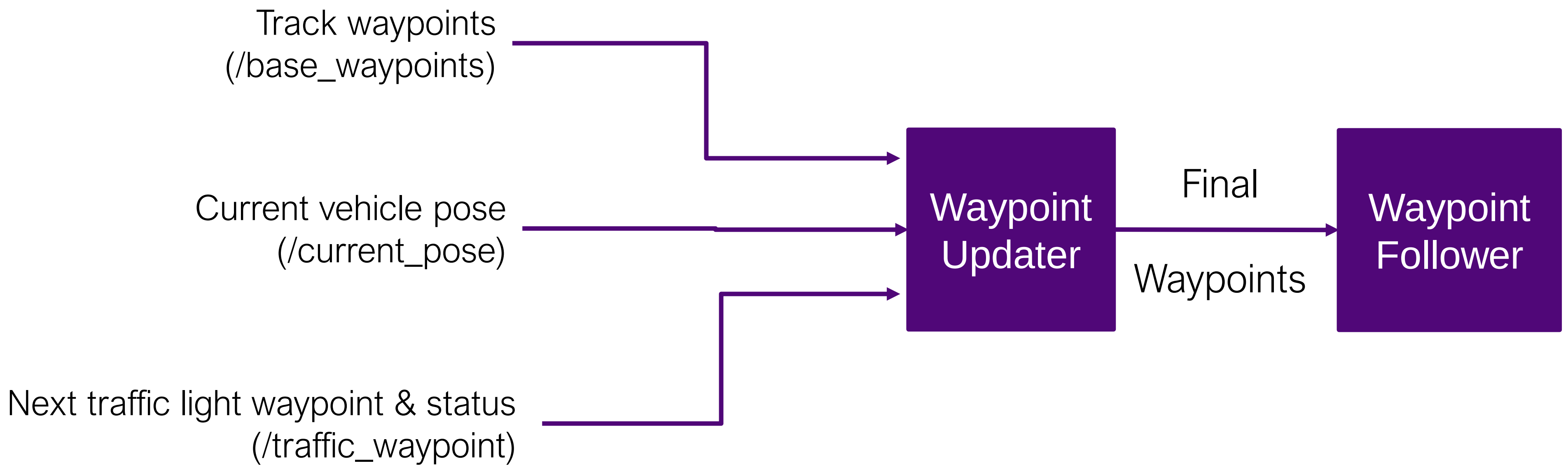


Car pose, track waypoints, traffic light positions
Simulator
traffic light states
Car pose & track waypoints
Current speed & angular velocity
Acceleration, brake torque and steering wheel angle
Traffic Light Detector
Next red light position
Waypoint Updater
Next N waypoints (position & speed)
Waypoint Follower
Target speed & angular velocity
Controller (DBW Node)
Perception
Planning
Control

Note: the "Simulator" is in fact composed of many nodes, but this should be irrelevant to your work

Waypoint Updater Node (1)

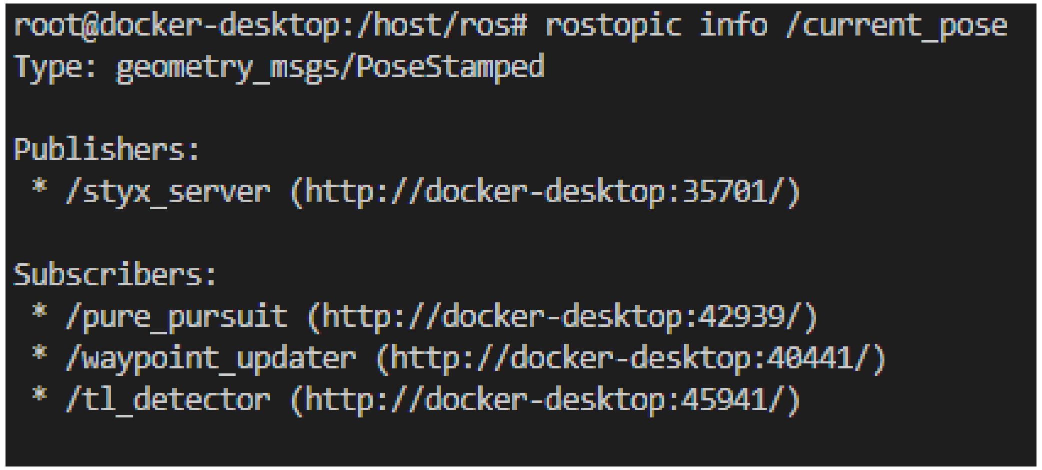
Goal: Publish a fixed number of waypoints *in front of the car* with correct target velocities



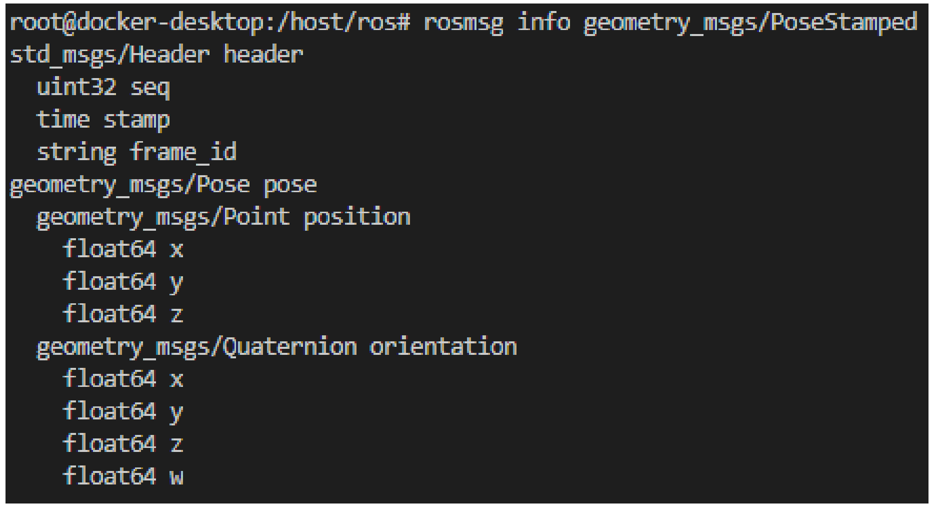
Waypoint Updater Node (1)

Getting information about the content of a message

- Run the simulator
- From the Docker container, run the code using **“launch.sh”**
- In another terminal tab, run **“rostopic info /current_pose”**
  - This will give you information about the topic, among which the message type
  - You can get all the details about a given message type using **”rosmsg info <message type>”**



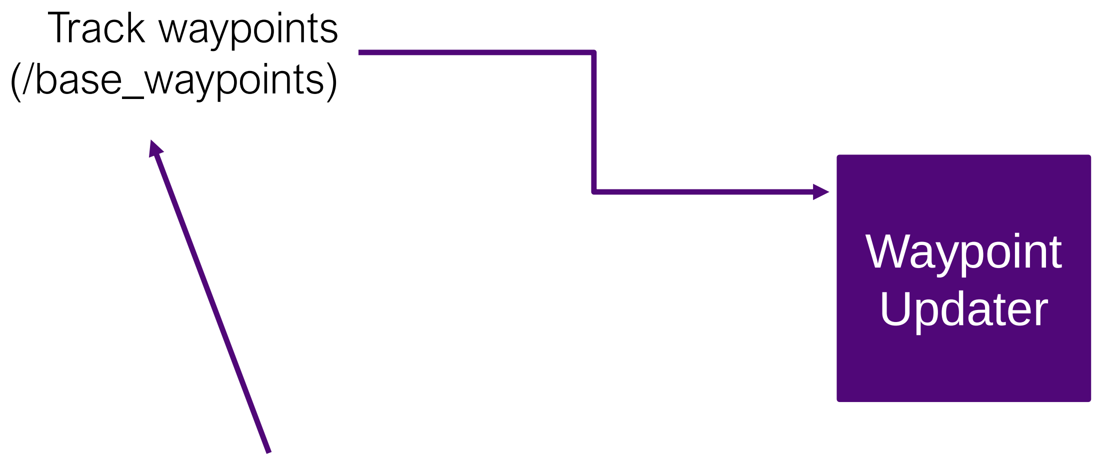
```
root@docker-desktop:/host/ros# rostopic info /current_pose
Type: geometry_msgs/PoseStamped

Publishers:
 * /styx_server (http://docker-desktop:35701/)

Subscribers:
 * /pure_pursuit (http://docker-desktop:42939/)
 * /waypoint_updater (http://docker-desktop:40441/)
 * /tl_detector (http://docker-desktop:45941/)
```



```
root@docker-desktop:/host/ros# rosmsg info geometry_msgs/PoseStamped
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

Waypoint Updater Node (1)

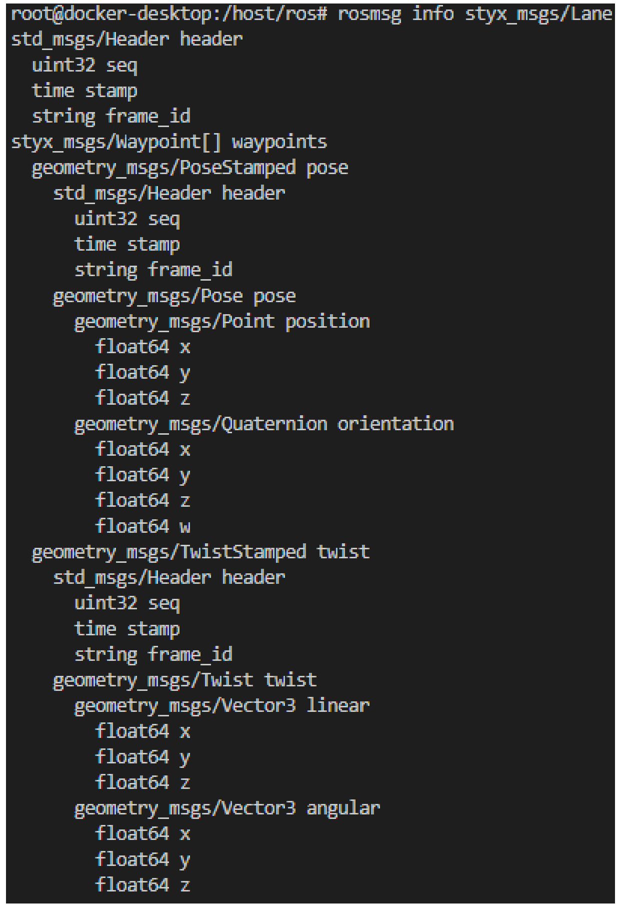
Goal: Publish a fixed number of waypoints *in front of the car* with correct target velocities



Messages contains a long list of waypoints, and for each waypoint:

- The position **`msg.waypoints[i].pose.position`**
- A target linear velocity **`msg.waypoints[i].twist.twist.x`**

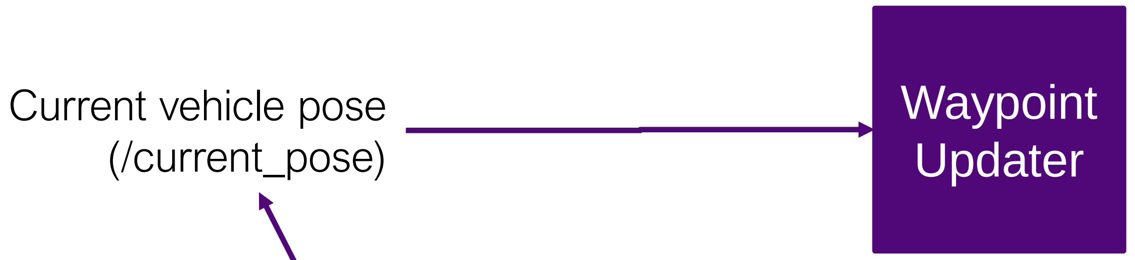
The rest of the information is irrelevant for your task
Note: the track waypoints only get sent once, since they never change

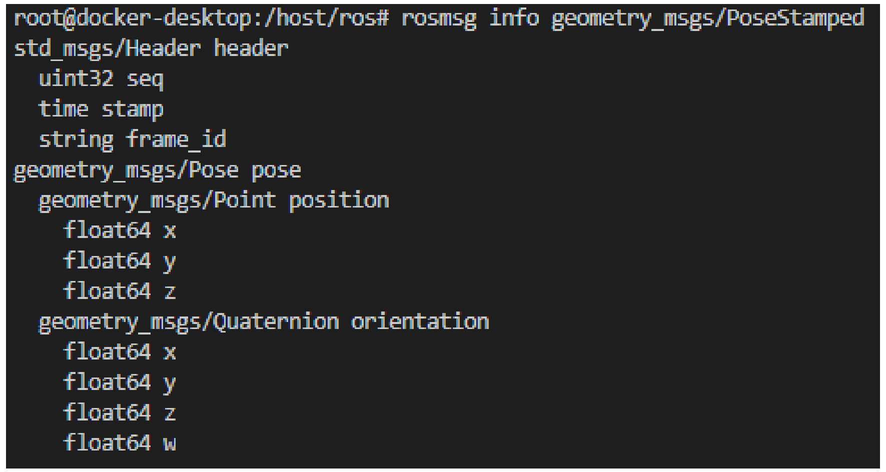


```
root@docker-desktop:/host/ros# rosmsg info styx_msgs/Lane
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
styx_msgs/Waypoint[] waypoints
  geometry_msgs/PoseStamped pose
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    geometry_msgs/Pose pose
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
  geometry_msgs/TwistStamped twist
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    geometry_msgs/Twist twist
      geometry_msgs/Vector3 linear
        float64 x
        float64 y
        float64 z
      geometry_msgs/Vector3 angular
        float64 x
        float64 y
        float64 z
```

Waypoint Updater Node (1)

Goal: Publish a fixed number of waypoints *in front of the car* with correct target velocities





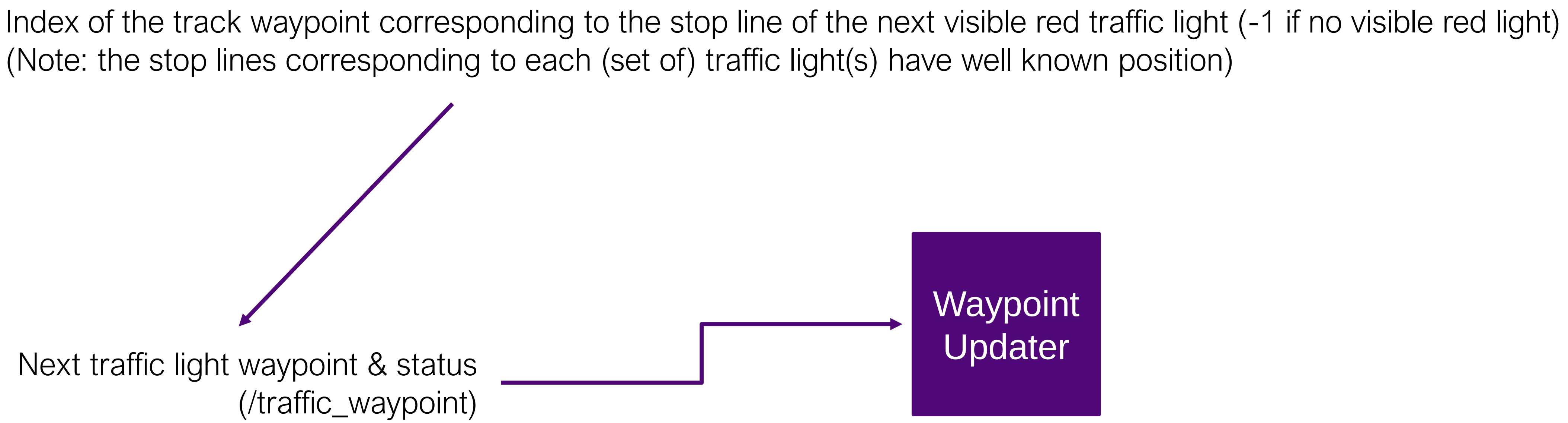
```
root@docker-desktop:/host/ros# rosmsg info geometry_msgs/PoseStamped
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

Exact pose of the car coming from the simulator: `msg.pose.position.{x,y,z}`

You may also access the orientation of the car expressed as a Quaternion

Waypoint Updater Node (1)

Goal: Publish a fixed number of waypoints *in front of the car* with correct target velocities


Index of the track waypoint corresponding to the stop line of the next visible red traffic light (-1 if no visible red light)
(Note: the stop lines corresponding to each (set of) traffic light(s) have well known position)
Next traffic light waypoint & status
(/traffic_waypoint)
Waypoint Updater

Waypoint Updater Node (1)

Advice: Don't try to do everything at once!

1. Start with something simple
   - Find the closest track waypoint **ahead** of the car's current (x,y) position (you can assume the track is flat and disregard z)
   - Output N waypoints from there on
   - Don't worry about efficiency or traffic lights yet

Assuming the car position is c and the closest waypoint is w_i, you will need to add a bit of math to figure out whether c is closer to $[w_{i-1}, w_i]$ or to $[w_i, w_{i+1}]$

→ One possibility: try to draw the situation and figure out what relation the vectors $c \rightarrow w_i$ and $w_{i-1} \rightarrow w_i$ should satisfy if w_i is ahead of c on the track

Waypoint Updater Node (1)

Advice: Don’t try to do everything at once!

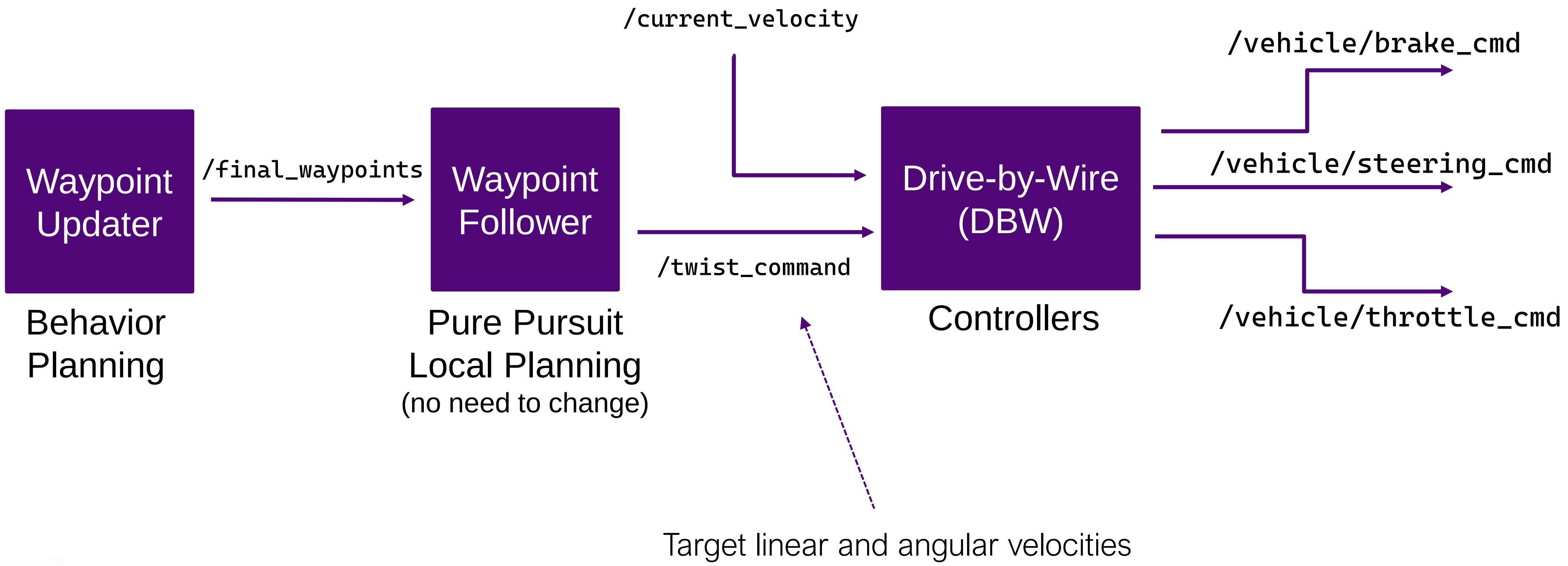
1. Start with something simple
   - Find the closest track waypoint **ahead** of the car’s current (x,y) position
   - Output N waypoints from there on
   - Don’t worry about efficiency or traffic lights yet

2. Work on other nodes until the car is able to drive around the track, completely ignoring traffic lights

3. Improve efficiency
   - You may need to improve efficiency before you proceed to other nodes if your computer is struggling
   - Finding the closest waypoint can be done easily in O(W) where W is the number of waypoints
   - This is of course sub-optimal, there are better solutions that exploit the fact that the track waypoints are fixed
   - *Hints:* See Closest pair of points problem and `scipy.spatial.KDTree`

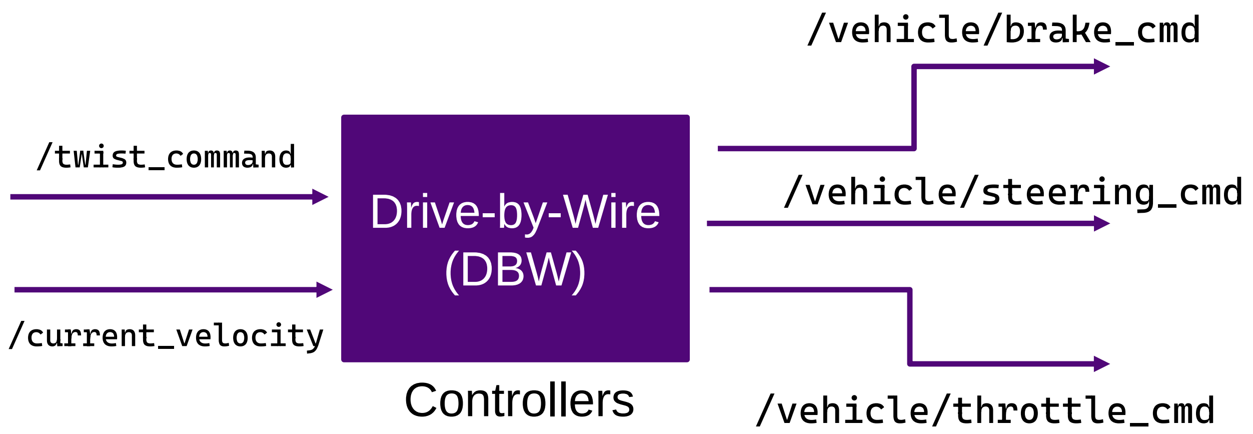
DBW Node (Controller)

Goal: Adjust motor control commands to match the target velocities



DBW Node

Goal: Adjust motor control commands to match the target velocities



- Throttle in [0,1]
- Brake torque in Newton-meters
- The steering is the steering wheel angle in radians, not the wheel angle, they are related through the steering ratio

DBW Node: Throttle/Brake Control

- Throttle: a number in [0,1], you should control it directly, and we suggest PID control
  - Tune your controller and also don't forget to clip the output to [0,x] where $x \leq 1$!

- Braking can be implemented as a "post-processing step" after you've implemented throttle
  We are modelling the car dynamics after a point mass, and physics says:
  - Braking torque is expressed in Newton-meters
  - Total Braking Torque = force applied * lever arm length = car mass * deceleration * lever arm
  - We are braking on the ground, but the force is applied on the wheel axle: lever arm = wheel radius!
  - Note that we're not computing torque per wheel, only total torque (don't multiply by 4)

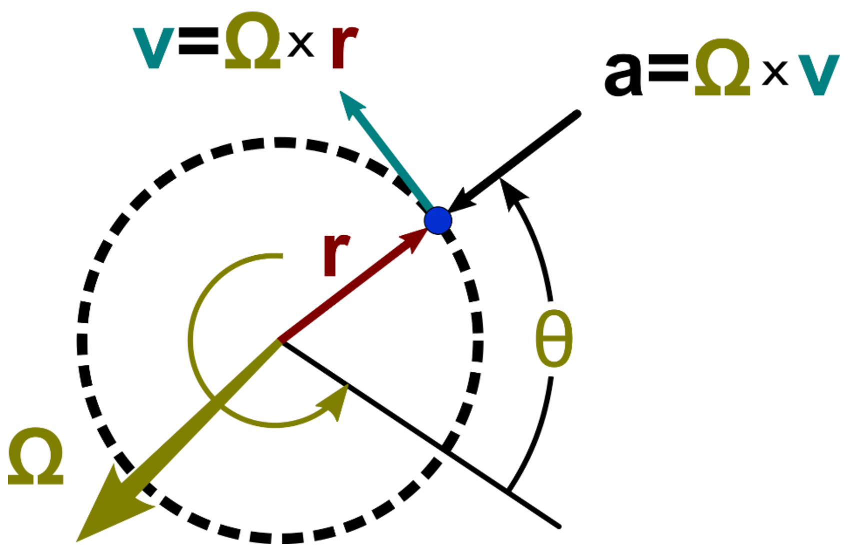
- A few observations
  - The car has an automatic transmission, so if target velocity is 0 and current linear speed is small ($\leq 0.1m/s$), you need to apply brakes ($\approx 700Nm$ needed so the simulated car won't move when it's at speed 0)
  - Don't brake and throttle at the same time!
  - Apply brake when throttle is already small, but you need to slow down more

- In order to get a smooth ride, you should also not decelerate more than the given deceleration limit

DBW Node: Steering Control

Goal: from current linear velocity v and target linear and angular velocities v^* and ω^*, we have to compute the appropriate steering angle θ for the vehicle

Assumption 1: Uniform Circular Motion



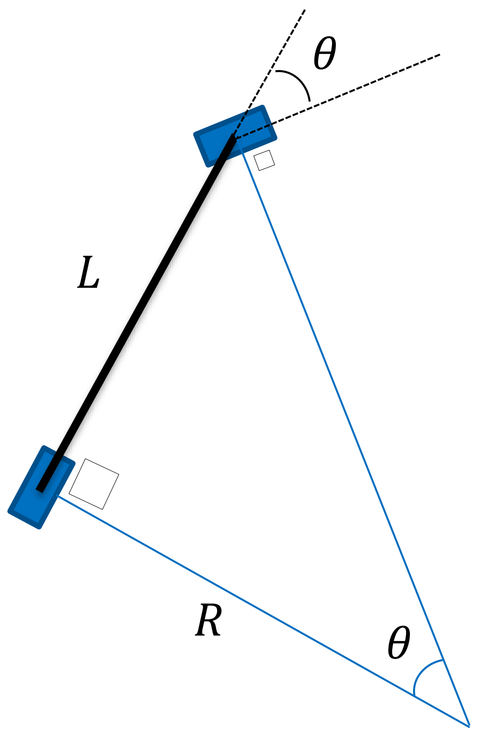
- Longitudinal speed is assumed constant
- Longitudinal acceleration has to be 0
- Acceleration is purely lateral (towards the center)
- Circular motion, by definition $v\ = \omega R$
- It can be shown that $a = \frac{v^2}{R} = v\omega$ (Proof)

Circular motion - Wikipedia

DBW Node: Steering Control

Goal: from current linear velocity v and target linear and angular velocities v^* and ω^*, we have to compute the appropriate steering angle θ for the vehicle

Assumption 2: Bicycle Model Kinematics



Instantaneous
Center of Rotation

Steering angle θ related to wheel base L and turning radius R

$$\tan\theta = \frac{L}{R}$$

DBW Node: Steering Control

Goal: from current linear velocity v and target linear and angular velocities v^* and ω^*, we have to compute the appropriate steering angle α for the vehicle

1. Compute angular velocity using Uniform circular motion equations:
   - Assume that that $\frac{v}{\omega} = \frac{v^*}{\omega^*}$
   - Cap $\omega \in [-\omega_{max}, \omega_{max}]$, where $\omega_{\max}$ is computed to never exceed the allowed max lateral acceleration

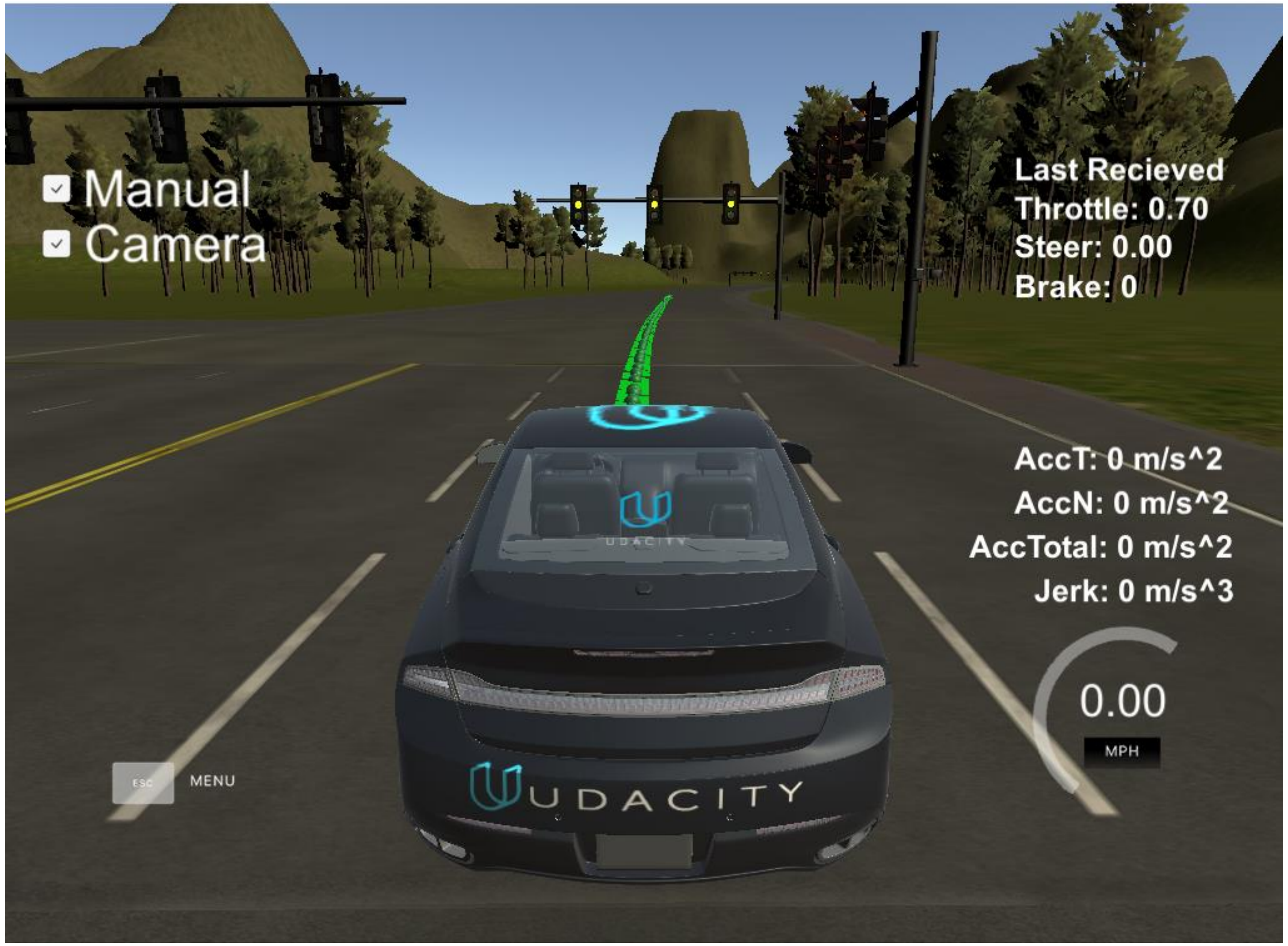
2. Compute the turning radius corresponding to ω

3. Use bicycle model equations to compute the steering angle θ from the turning radius value

4. Compute the steering wheel angle $\alpha = steering\ ratio \times \theta$ and cap $\alpha \in [-\alpha_{max}, \alpha_{max}]$

After Correctly Implementing these Two…

You car should now be driving along the track and completely ignore traffic lights



Last Recieved
Throttle: 0.70
Steer: 0.00
Brake: 0

AccT: 0 m/s^2
AccN: 0 m/s^2
AccTotal: 0 m/s^2
Jerk: 0 m/s^3

0.00

Traffic Light Detector (1)

Basic Traffic Light Detection: Use the ground truth from the simulator

- You don’t need to have a working traffic light detector node yet!
- The ground truth for state and position of all traffic lights of the simulator can be read from **`/vehicle/traffic_lights`**, so use that to debug your implementation, and use **`/traffic_lights`** (output of traffic light detector) once things work with the ground truth!
- This part is provided for you, but you will need to understand the implementation to implement a Traffic Light Detector

Waypoint Updater Node (2)

Second round of updates for Waypoint Updater: Implement traffic light mechanics

You now need to worry about setting adequate target velocities on the waypoint you are outputting so that the car can *smoothly decelerate and stop near the traffic light stop line*

- Be careful not to decelerate more than the maximal allowed deceleration of 0.5
- You are free to choose a "deceleration curve", *e.g.* it could be linear from current speed to 0, but something like a quadratic function might look more like what humans do (brake slow first and harder when near the stop line)
- Make sure to test these updates with the ground truth traffic light states before you use your own traffic light detector

Your car should now gracefully stop near traffic light stop lines when light is red

Tip: To enable fast testing, light is always red when the simulator starts

Traffic Light Detector (2)

Traffic Light Detection: Implement a working traffic light detector (25% of grade)

- Need to train to perform classification (good) or object detection (even better)

- You have complete freedom on the design of this part of the project

- Keep the computational resource constraints in mind!
  - You'll be running inference on the CPU, so use a lightweight network
  - Resize the input so that resolution is not too high but traffic lights are still clearly visible
  - Many object detection architecture have a variant optimized for speed (eg. YOLOv3 Tiny)

- Training your model
  - Data: You may reuse an existing labeled dataset shipped with the project (see `tl_detector` folder) or capture your own data from the simulator if you are familiar with ROS (rosbags)
  - You should train your own classifier on the Google Colab infrastructure and only load it the project
  - Your final training code is part of the project deliverables and should be included in your sources
  - You may use either Tensorflow (2.3.0) or PyTorch (1.6.0) to run your classifier

Office Hours

Office Hours

- We have made a lot of effort to prepare a Docker environment that should work on all platforms
  - This is the 1st iteration of the project, so there could be some bumps along the way
  - Try to run the software environment as soon as possible so we can detect issues early
  - There will be on-demand office hours available in January
  - In the meantime, if you experience technical difficulties with the software environment or something in the project description is unclear, send me an email
  - We will maintain a FAQ document on the project's repository

- If your computer is not fast enough to run the simulator and the software, there are several options
  - Run the simulator in the lowest graphics settings
  - Decrease the frequency of ROS loops
  - If this is still a big problem, reach out and we will try to figure out a solution

Conclusion

GREAT COURSES ARE BUILT ON FEEDBACK

HAVE YOUR SAY ON **FEEDBACK.UNI.LU**

FROM 14 DECEMBER TO 10 JANUARY

WE BUILD THE UNIVERSITY TOGETHER.

uni.lu UNIVERSITÉ DU LUXEMBOURG

You may Scan the QR Code

or

Go directly to: https://feedback.uni.lu

- Log in using your University credentials.
- Choose your preferred language.
- Answer the questions thoughtfully and constructively.

For questions, please contact course.feedback@uni.lu

Thank you!

Flash Me !

Winter Semester 2020
14 December – 10 January

UNIVERSITÉ DU LUXEMBOURG

The Recruiting Slide

- If you have a good track record, we will have some openings to work on your master thesis in the 2nd semester

- If you work with François, the topic will be related to unsupervised, self-supervised or weakly supervised learning for perception in autonomous vehicles
  - Exact topic to be discussed and defined
  - Not very restrictive: there are tons of possible applications (depth estimation, segmentation, free-space detection, …)

- Don't wait to reach out, we should start outlining the topics soon!
- Not only Deep Learning, the lab works on many different topics
  → slides from 1st lecture for more infos

Interdisciplinary Centre for
Security, Reliability and Trust

Thank You For Your Attention!

François Robinet
francois.robinet@uni.lu
360lab.uni.lu