ENPH 353 2021

UBC Parking - competition notes

1. Competition overview

The competition takes place in the simulated environment presented in Fig. 1. Your goal is to develop an autonomous agent that drives through the environment while obeying traffic laws and returns license plates and associated parking IDs.

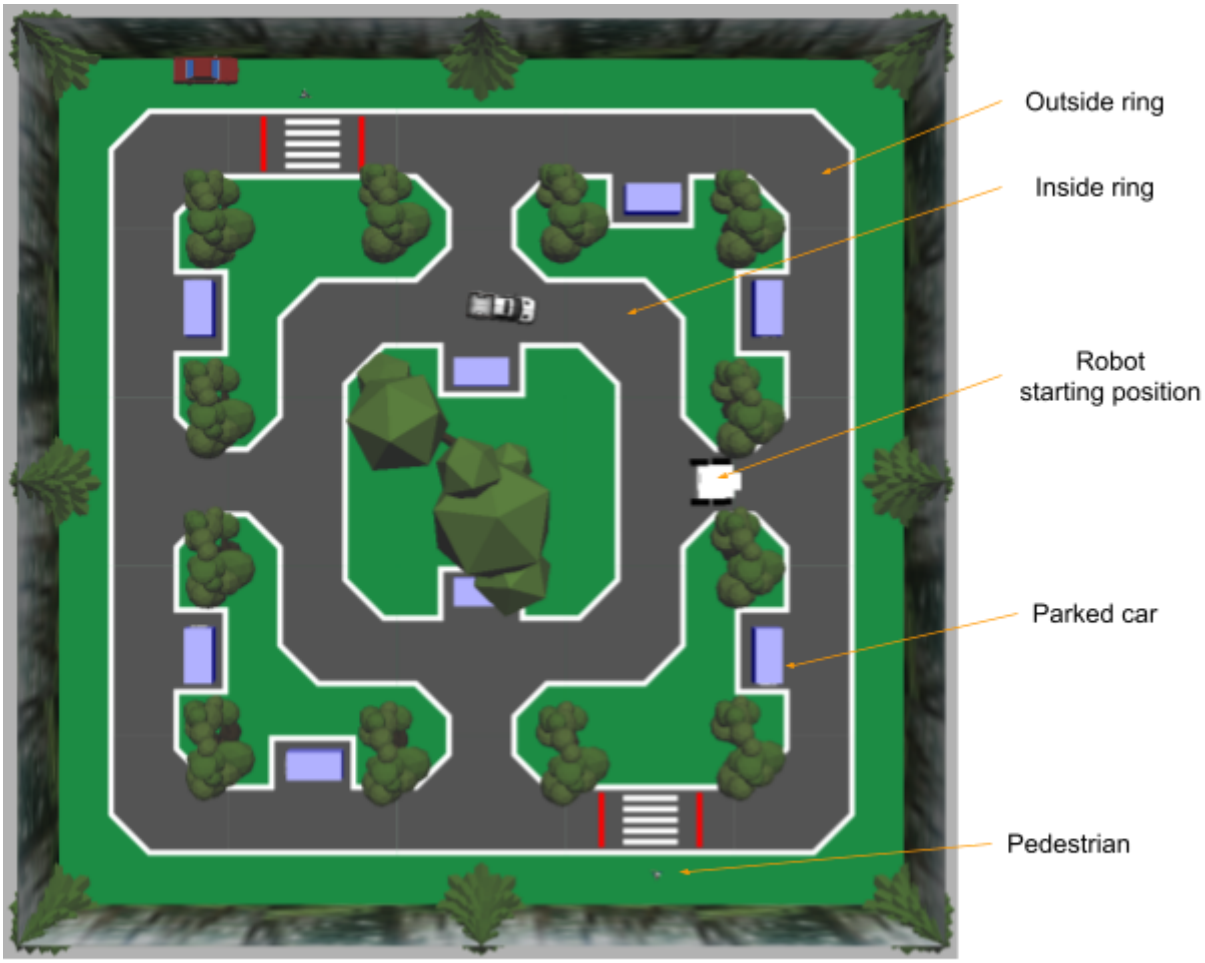


Fig. 1 Simulation playing arena overhead view

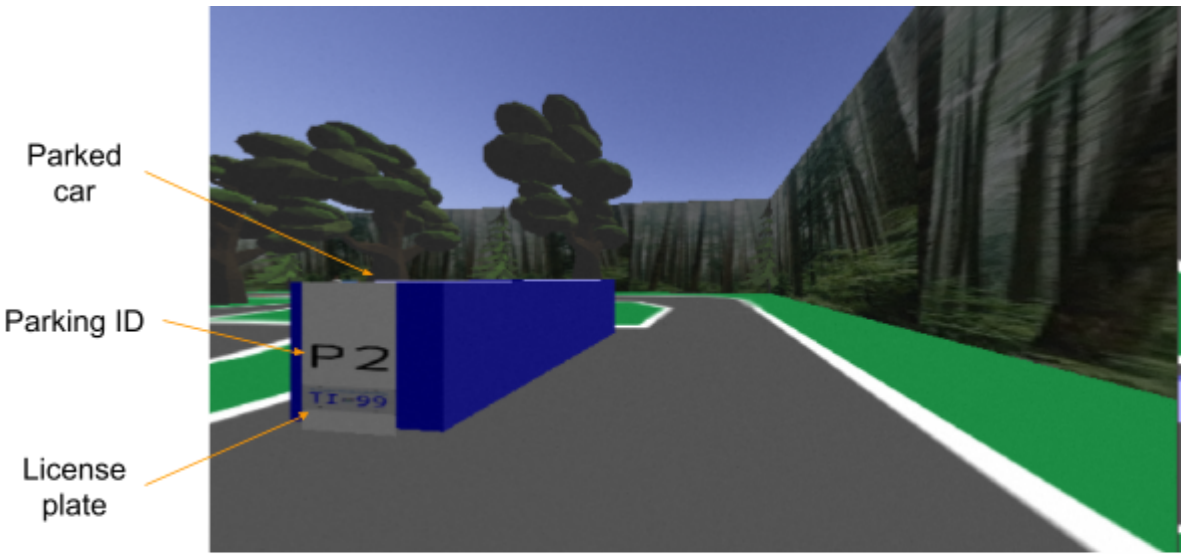


Fig. 2 In simulation camera view from robot
(used as input for team controller)

2. Competition rules and scoring

Table last updated: 2021-02-01 (We)

1. Players	1. A team consists of 2 students 2. Each team controls one robot
2. Goal	1. Each team participates in a 4 minute (sim-time) round where they return license plates and locations 2. For each correct license plate and location returned the team is awarded points 3. Points are deducted for Traffic Rules violations 4. The team with the highest score wins
3. Robot control	1. The competition is taking place in simulation (Gazebo) 2. A team is provided with a live image feed from the in-simulation camera mounted on top of the robot 3. The ROS node the team develops can send control commands to move the robot backwards/forwards, turn it CW/CCW or stop it 4. A team can submit a message with (license plate id, location id) to ROS Master Correct plate id: +6pts (outside ring), +8pts (inside ring) 5. During time-trials and competition your control algorithm is allowed to use only the following topics from the simulation: - **Subscribe** to: /R1/pi_camera/image_raw topic - **Subscribe** to: /clock - **Publish** to: /R1/cmd_vel topic and /license_plate topic 6. You can publish and subscribe to as many topics as you need to if these are topics you created on your end of the system (i.e. the topics are not originating from or ending at the simulation).
4. Traffic Rules	1. Maintain at least 2 wheels inside the white lines marking the road (-2 pts) 2. Do not collide with other robots (-5pts) NOTE: If the NPC truck rear-ends your robot no penalties are given 3. Do not run over pedestrians (-10pts) 4. Complete a full outside ring lap (+5pts)
5. Start/stop competition	1. To start the timer for the competition send a message on the /license_plate topic with license plate ID 0 (zero) i.e. str('TeamRed,multi21,0,XR58') 2. To stop the timer for the competition send a message to the /license_plate topic with license plate ID -1 (minus one) i.e. str('TeamRed,multi12,-1,XR58')
6. Other rules	1. You must design, train and use your own neural network. You cannot use pre-trained neural networks (i.e. Tesseract)

2.1 Time trials marking

During time trials we will award the 5% marks on a pass/fail basis. The pass criteria are to have a ROS package that:

- starts the timer
- once the timer is started your robot moves at least 1 meter in the competition world (each “square grid” in the competition is 1 meter).
- stop the timer

These are the minimum requirements - we hope / expect to see much more functionality implemented.

A few more details:

1) During time trials only one team will be on Zoom at a given time. We set-up a schedule for each team to demo their robot's performance. Please find your time slot here. Both me and Josh will be on Discord throughout the time trials if you need to ask questions (though we might lag somewhat in our replies).

2) We will ask you to demo the robot's performance live. However if you cannot do this please have a video link available with a pre-recorded run. This video should be recorded in the last 60 minutes before your allotted time. As a bit of an aside we will use the outcome of the time trials to determine if we can run our final competition live or we need to pre-record it.

3) When you are ready to demo please set-up at least a 3-pane window scheme on your screen for both the live performances or the pre-recorded ones. One window should be a terminal window, one should be the Gazebo simulator and one window will be the score tracker. In addition to these windows you can display any other troubleshooting windows you have available.

4) In the terminal window please run 3 tabs (to open a new tab in the current window press Ctrl+Shift+T).
   - Terminal 1: runs the simulation (i.e. ./run_sim -pgv)
   - Terminal 2: runs the score tracker
   - Terminal 3: runs your custom controller

5) Please make sure you have the latest version of the simulation pulled from github. And ensure you don't have your custom controller script running from the competition package (see section 4.1 of the Competition Notes).

3. Competition environment

3.1 ROS environment set-up

To set-up your environment for the competition follow these steps:

1) If you **do not** have a ROS workspace already created in your home directory, make one - and change directory to the src folder in it.

```
mkdir -p ~/ros_ws/src
cd ~/ros_ws/src
```

2) Clone the competition environment:

```
git clone https://github.com/ENPH353/2020_competition.git
```

3) Build the environment:

```
cd ~/ros_ws
catkin_make
```

4) Source the environment:

```
source ~/ros_ws/devel/setup.bash
```

5) Start the simulated world:

```
cd ~/ros_ws/src/2020_competition/enph353/enph353_utils/scripts
./run_sim.sh -vpg
```

The available flags for the simulation are:

Option	Description
-v	spawn vehicle
-p	spawn pedestrians
-g	generate new license plates

6) Start the score tracking app Open a new tab in the current terminal window by pressing Ctrl+Shift+T
The new tab will open in: `~/ros_ws/src/2020T1_competition/enph353/enph353_utils/scripts`
Launch the score tracking app:

```
./score_tracker.py
```

3.2 ROS inputs and outputs

During time-trials and competition your control algorithm is allowed to use only the following topics from the simulation:

- **Subscribe** to: /R1/pi_camera/image_raw topic
- **Subscribe** to: /clock
- **Publish** to: /R1/cmd_vel topic and /license_plate topic

You can publish and subscribe to as many topics as you need to if these are topics you created on your end of the system.

3.3 Score tracking app

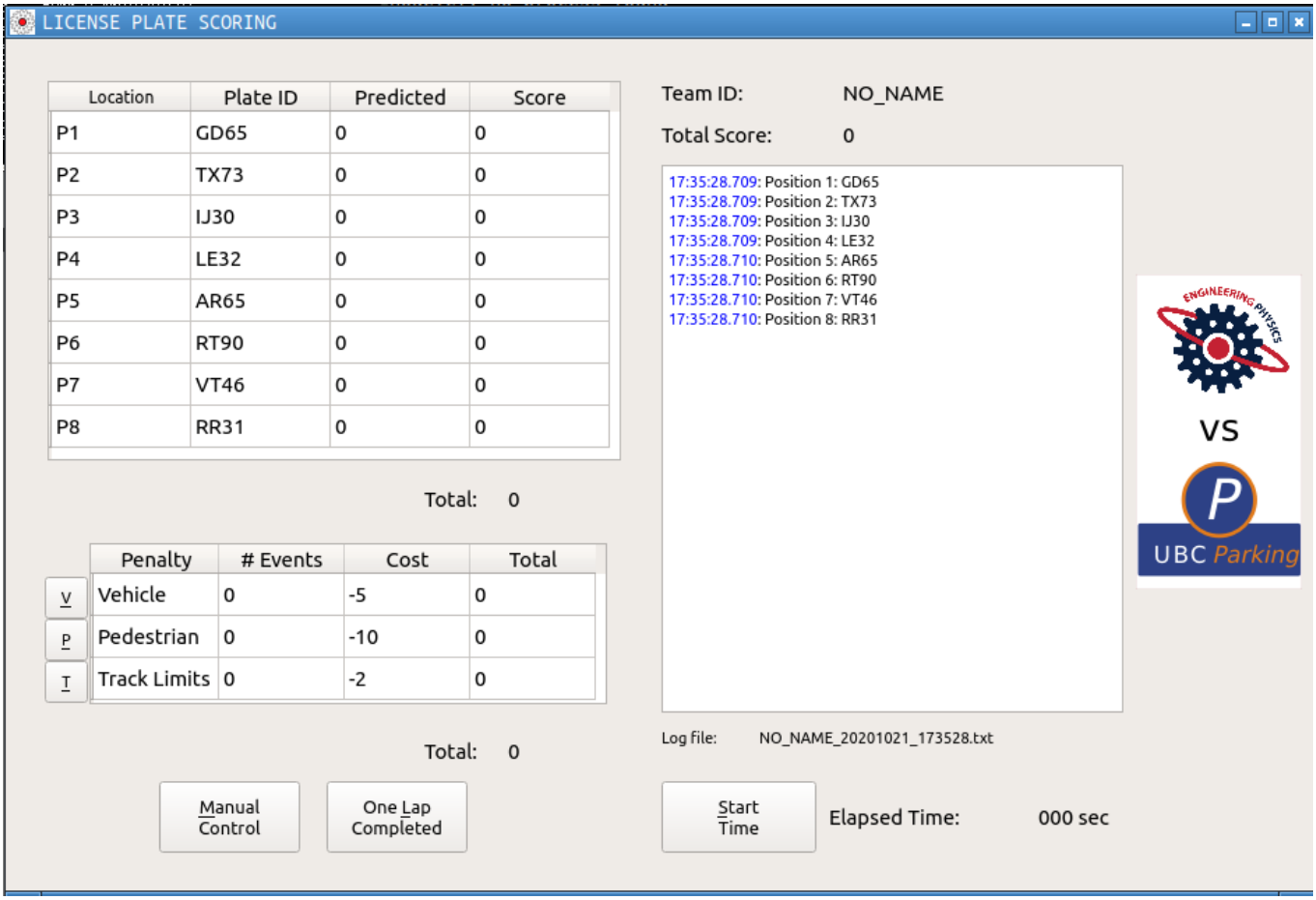
The topic name and format for the message you use to report the license plate location and ID are
Topic: /license_plate
Message type: std_msgs/String

Note: To publish a string on topic /license_plate you will need to add the following import into your python script:
`from std_msgs.msg import String`

The string you send must contain the following comma separated values:
team ID: max 8 characters (no spaces)
team password: max 8 characters (no spaces)
license plate location: int (0 to 8); 0 is a special case - see below
license plate id: 4 characters (no spaces)

example: str('TeamRed,multi21,4,XR58')

To help you test reporting license plates we added a scoring system script (score_tracker.py) in the same folder as ./run_sim.sh. To run the scoring system first start the competition simulation and then in a separate terminal start the score_tracker.py. Once the score tracker is started a /license_plate topic is added to the ROS competition instance and you can publish to it.



Score Tracker UI

NOTE: The first message your robot must publish must use license plate location 0. This message is a special message used to start the timer and register your team with the system. Similarly the last message your robot must publish must use license plate location -1. This message is also a special message used to stop the timer. Neither of these messages are awarded any points.

The scoring system logs to file all the messages you submit and the awarded scores. Make sure to delete these files if there are too many of them - they are located in:
`$ ~/ros_ws/src/2020T1_competition/enph353/enph353_utils/scripts`

3.4 ROS Cheatsheet

Command	Description
rostopic list	list all the topics available in this ROS instance
rostopic echo <topic_name>	watch values sent on a particular topic
rostopic pub <topic_name> <message>	send values on a particular topic
rosrun rqt_image_view rqt_image_view	use to display the camera feed
rosrun rqt_graph rqt_graph	display the ROS system's node connectivity
rospack profile	ROS doesn't always detect all the packages we have created. Run this command to crawl all directories and identify manifest files for our projects.

3.4.1 Creating a new ROS package

In your workspace src folder run (i.e. ~/ros_ws/src):
`$ catkin_create_pkg <pkg_name> <pkg_dependencies>`
example:
`$ catkin_create_pkg my_controller rospy`

after this you will need to rebuild your workspace:
`$ cd ~/ros_ws`
`$ catkin_make`

3.4.2 Restarting the simulation

To stop the simulation most often we will use `Ctrl + C` to terminate the Gazebo process.

Sometimes Gazebo does not shutdown elegantly and we have to run `killgazebogym` from a new terminal.

As we are running multiple nodes in the simulation I also suggest using `pkill -f -9 "2020T1_competition"` to shutdown any nodes associated with the competition simulation. This last command kills any process that contains the string "2020T1_competition" in its path.

3.4.3 Timing details

After you create your Publisher and Subscriber objects in your custom controller you should add a delay of 1 second before you send any messages. This allows the ROS master node to complete the registration procedure and get ready to process communication.

Do not spam messages on topics that have a queue set to 1. That is you should wait 100-200ms before you send a message so you allow for the previous message to be processed. Otherwise some messages will get dropped.

3.5 Tensorflow troubleshooting

3.5.1 Computational graph error

Some of you will encounter a cryptic error about not being able to use the computational graph of the model you have trained when trying to run predictions from ROS. The solution (as detailed here) is to:

1) Import tensorflow libraries

```
import tensorflow as tf
from tensorflow.keras import models
from tensorflow.python.keras.backend import set_session
from tensorflow.python.keras.models import load_model
```

2) Create a global instance of our tensorflow session and computational graph (i.e. at the top of our file outside any class or function call).

```
sess1 = tf.Session()
graph1 = tf.get_default_graph()
set_session(sess1)
```

3) Then load our model (normally as part of a class):

```
self.plate_NN = models.load_model(plate_NN_path)
```

4) Do our prediction using the global session and computational graph:

```
global sess1
global graph1
with graph1.as_default():
   set_session(sess1)
   NN_prediction = self.plate_NN.predict(char_aug)[0]
```

3.5.2 Saving models in Colaboratory

Because we are still using tensorflow 1 when you save your models you might need to use the older version of the h5 package. To do this install h5py version 2.10.0 in the colab notebook before saving the model. You can do this by running:

```
!pip install h5py==2.10.0 --force-reinstall
```

4. Notes on implementing an agent

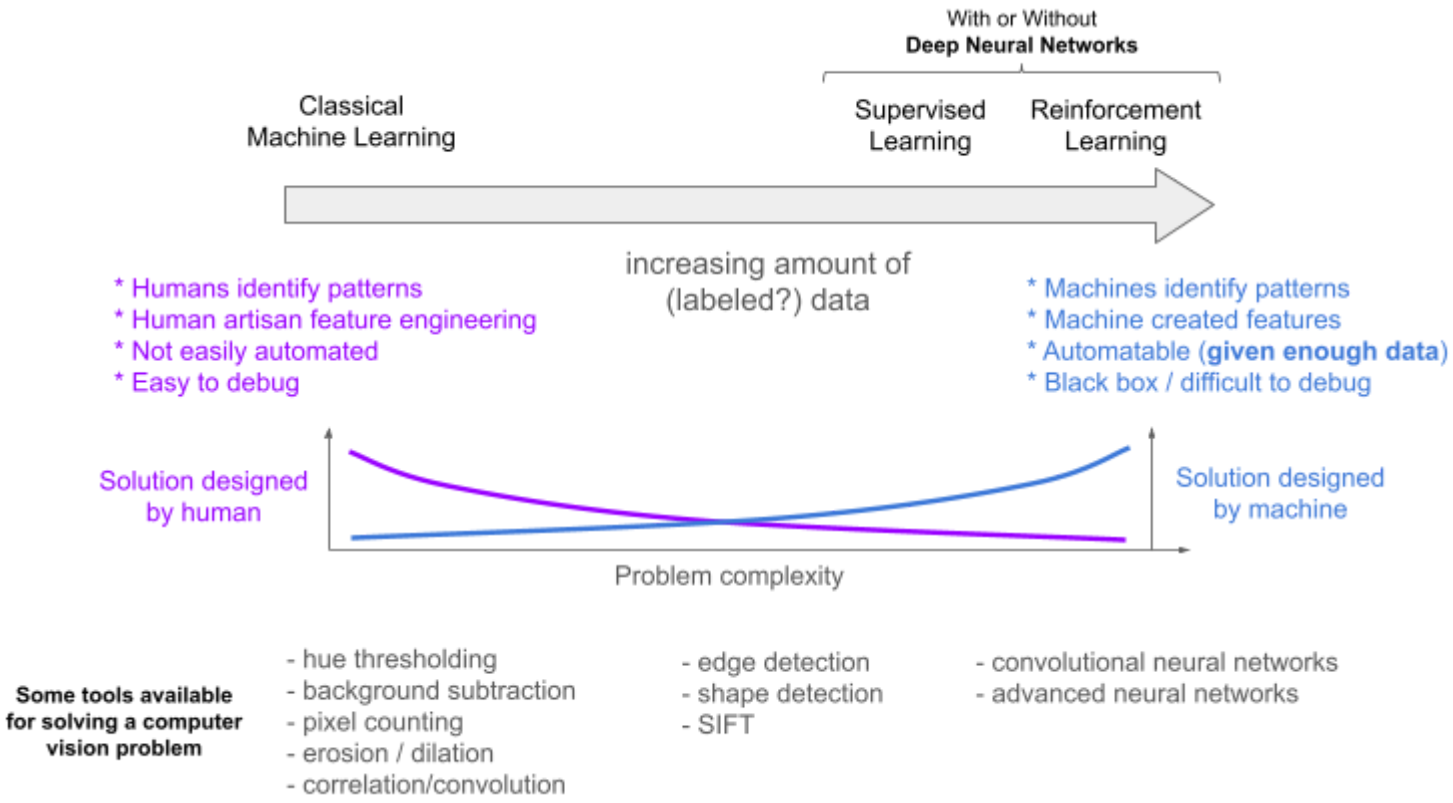
At a minimum implement:

- lane following
    - You can use any technique you think is appropriate. A few options are (in order of difficulty):
        - Classical computer vision with PID
        - Imitation learning (see End-to-end learning for self-driving cars paper)

        - Reinforcement learning Q-learning
        - Reinforcement learning DQN
- license plate reader
    - license plate detector
    - license plate character recognition (most likely a CNN)
- pedestrian and vehicle detection
- create your own debugging video screen so you understand how the robot interprets the word.

You can use any technique you want in the competition. You are not limited to neural networks.

When solving a problem related to your competition controller, stop, take a step back and breath. Before you dive into the details of the solution, scan the horizon of possible approaches and choose one that balances the complexity of the problem with the complexity of the solution. The diagram below offers you a simple heuristic for what is available for a problem related to computer vision.



Also remember not to over-engineer yourself out of time and energy - “the best solution is to not have the problem”. In more applicable terms aim for simple solutions.

NOTE A: unless it is absolutely necessary do not run your ROS applications through VS Code’s terminal. Instead I strongly suggest you use a standard terminal to debug issues. Good alternatives to VS Code are Sublime and Atom.

NOTE B: To select colour thresholds using HSV you can use this script.

4.1 Software structure and source control

In general there are two methods of doing source control for our project. We can either place all our packages in a single large repository. Or we can store each package in its own repository. To determine which of these configurations we should use we need to consider how much coupling there is between the functionality of the packages we will do source control on. If some packages are tightly coupled, that is if when we change something in one package we need to update the others, we should use a single repository for all packages. Otherwise we use independent repositories for each package. Structuring our source control based on the amount of coupling between packages will reduce the amount of updates we need to keep track of and also allows us to simplify our integration testing. On the other hand having a repository for each package will, by design, maintain the modularity of our software which will improve the reusability and robustness of each section of code. At some level of complexity deciding which structure to adopt becomes a matter of personal choice.

For our competition one software structure I can see working is to have a few repositories like below:

- [repo 1] The competition_pkg repository (cloned from ENPH353 2020 UBC Parking Competition) is a ROS package containing:
    - Competition world and robot
    - Scoring system

- [repo 2] The controller_pkg repository is what each team will create. This is a ROS package containing:
    - Control algorithm for the robot
    - Potentially also includes a convolution neural network model or a reinforcement learning policy file trained from [repo 3] and [repo 4] respectively

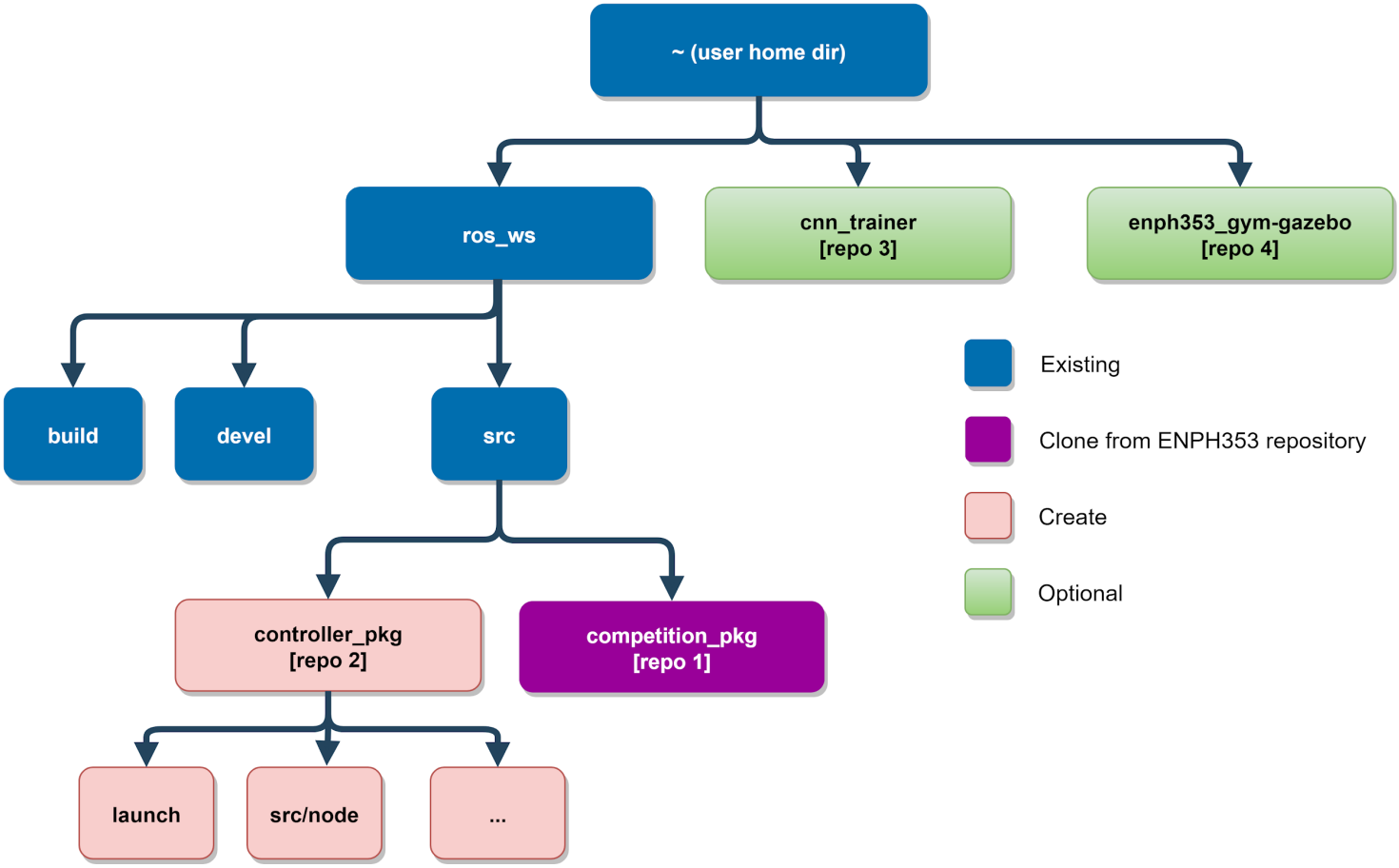
- [repo 3] Convolutional neural network repository contains
    - Data generation scripts
    - Training data
    - CNN training code

- [repo 4] Reinforcement learning gym repository contains
    - Competition world and robot (copy the ENPH353 2020T1 UBC Parking Competition)

You will need to remove the .git folder that is part of the 2020 UBC Parking Competition repository so that you will not end up with a git repository inside a git repository.

    - Reinforcement learning gym framework

The following diagram is a suggested folder tree structure for the competition



4.2 Agent training using Gym-gazebo

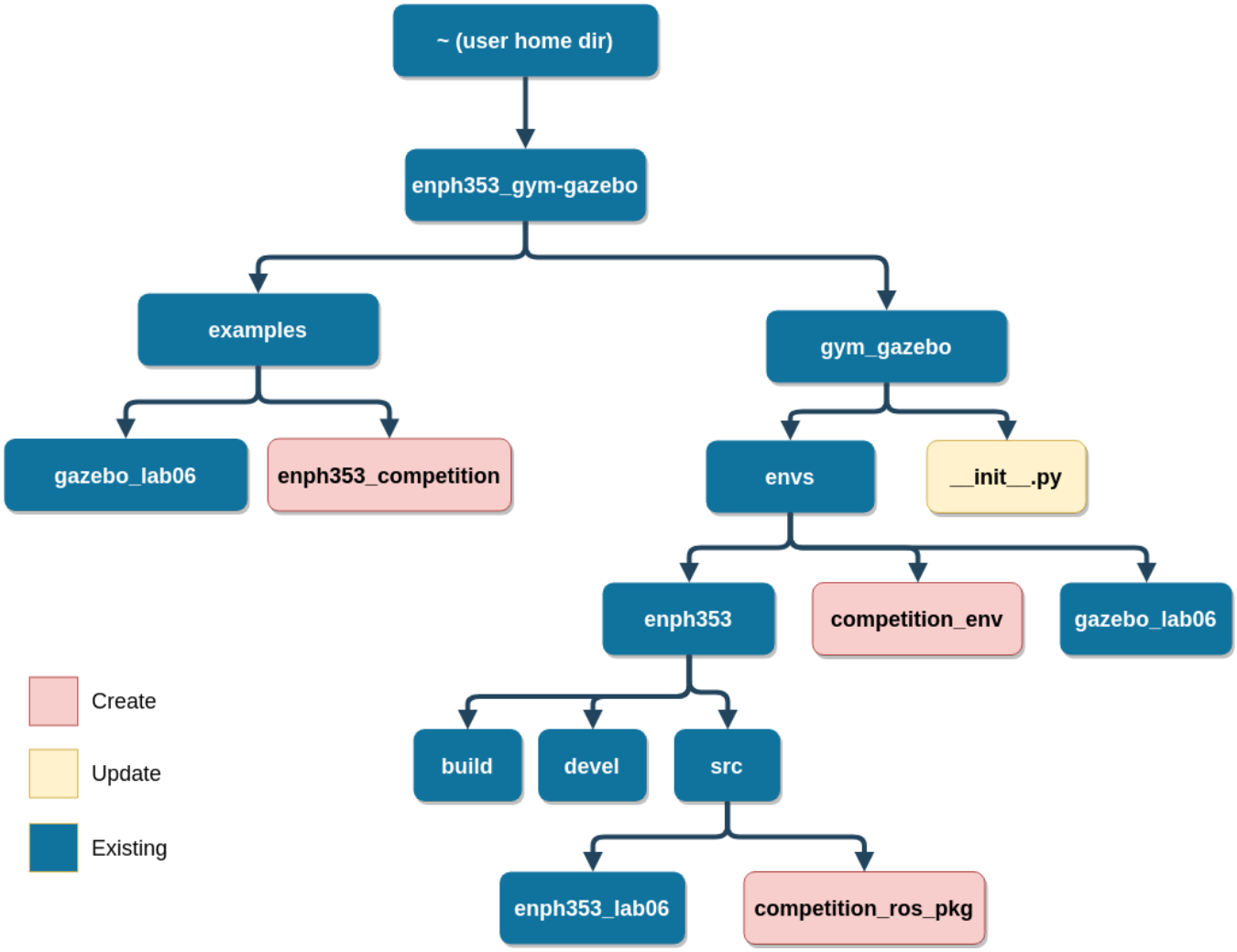
If you want to use gym-gazebo to train an agent to interact in the competition simulation ensure you use gym-gazebo properly.

Like its ‘gym’ name implies, gym-gazebo is a place where you train an agent. Once the agent is trained you no longer need to use the gym-gazebo framework. That is when you go into competition you don’t run your code from the gym-gazebo folder structure you run it from a dedicated competition workspace.

Let us say we want to use reinforcement learning to train an agent to perform lane keeping for our competition. The process we would follow is:

1) Create a training environment in gym-gazebo that contains the competition robot and world (you can copy the world and robot from the ROS competition package to create this new environment).
2) Determine what the state space is and code the image processing that determines the current state.
3) Establish what the available actions are, the reward function, exploration/exploitation rate, discount factor, learning rate, etc (most of these you will obtain from experience - educated guesses).
4) Train agent and save the policy that works best.
5) Use the policy in your actual competition code.

To create a new training environment and to train an agent you will need to add a few new folders and files to the gym-gazebo directory structure. These modifications mirror what we did in lab six and are summarized below:


~ (user home dir)
enph353_gym-gazebo
examples
gym_gazebo
gazebo_lab06
enph353_competition
envs
__init__.py
enph353
competition_env
gazebo_lab06
build
devel
src
enph353_lab06
competition_ros_pkg
Create
Update
Existing