Project Deep Learning

Senior Design Team 10 (SD_MAY_10)

The Team

Jazzlyn Jacobus - Computer Engineering

Benito Moeckly - Computer Engineering

Jose Carlos Garcia - Computer Engineering & Cybersecurity Engineering

Caleb DeBoef - Electrical Engineering

Advisor & Client: Professor Rover

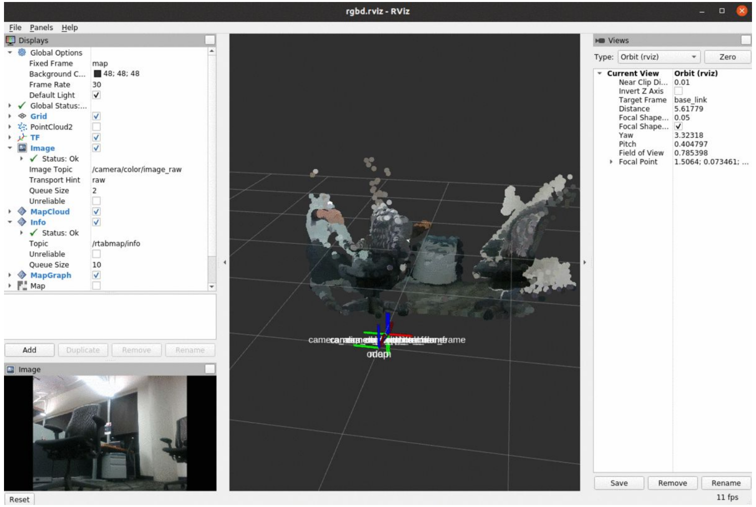
Project Vision

Problem Statement

Machine Learning Engineer 4th fastest growing job market (via LinkedIn)

The gap in Iowa State's ECPE curriculum

- No dedicated classes on machine learning for embedded systems for undergraduates
- Machine learning is a quickly growing field, Iowa State should update the core curriculum accordingly



Our (initial) solution

Initially, we explored the possibility of developing a CPRE 288 lab involving the AWS DeepRacer (more on this later) and teaching the robot to run a few laps via reinforcement training

Teaching the AWS DeepRacer turned out to be trivial, among a few other issues:

- Using the AWS servers can get expensive quickly
- The robot's track is space consuming
- AWS only offered access to the reward function

Image courtesy of aws.amazon.com

The Deep Learning Solution

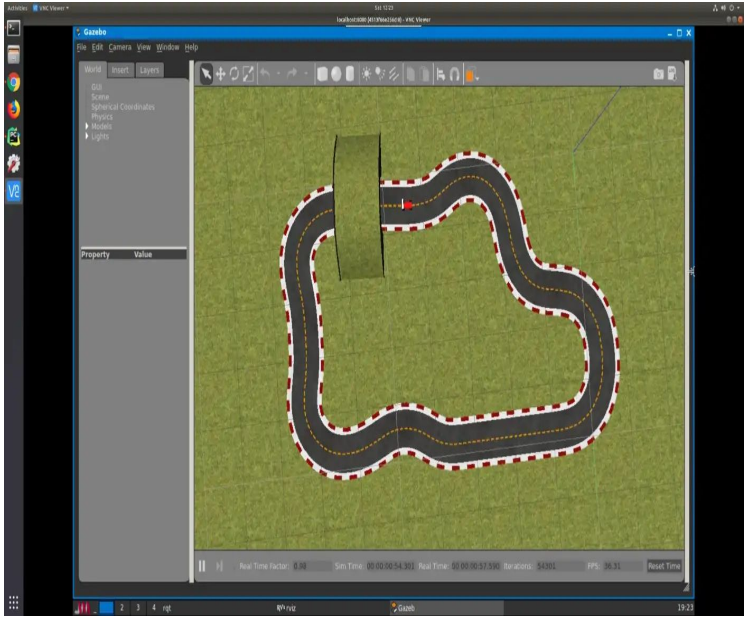
A testbed built on the DeepRacer robot

The DeepRacer software went open source in April of 2021

Students will learn about embedded machine learning through a custom DeepRacer Core Application via labs/assignments

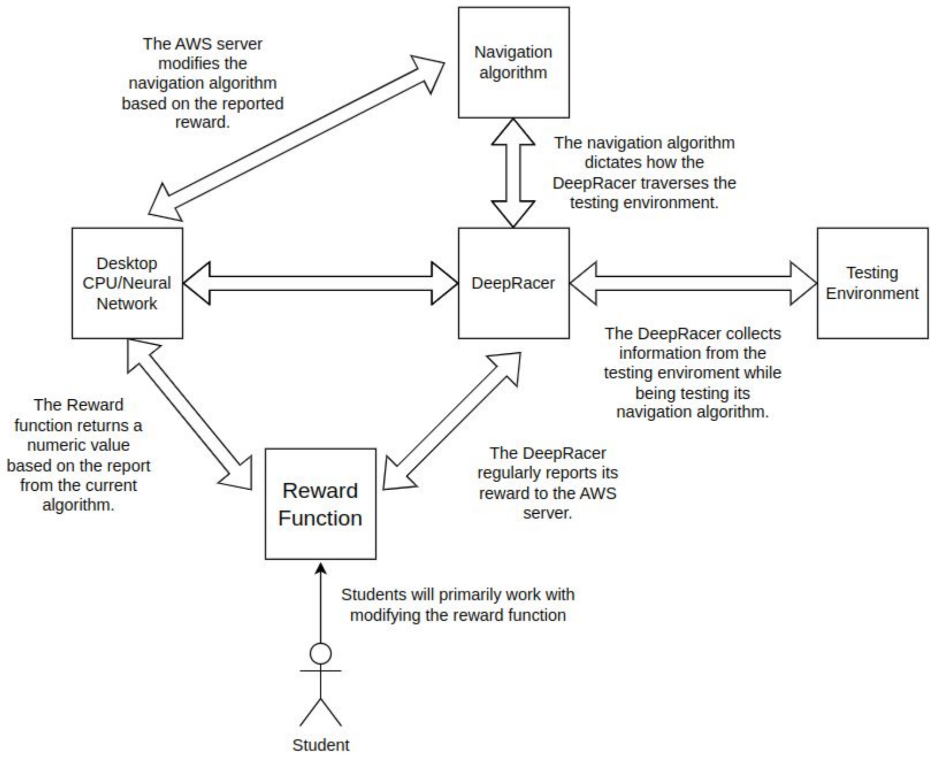
Particular focus on navigating the maze from CPRE 288: Embedded Systems with ML

Students interact with DeepLearner via desktop interface



Overview of “DeepLearning”

AWS Deepracer Top-level System Diagram



Built for the student

Focus on basic machine learning

Allow for parameters to be passed to the sensors on the DeepRacer

Allow students to build their own reward functions based on sensor input

Students will be able to choose from Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) training algorithms

Requirements

User Requirements:

- Must introduce Machine Learning topics in an easy to understand environment for students based in Embedded Systems
- Provide students with the ability to easily change sensor parameters and submit their reward function

Software Requirements:

- Must implement a locally trained model for scalability and to reduce costs
- Must provide the ability to be simulated and showcased virtually

Lab Requirements:

- Develop a lab based on CPRE 288 Maze project using ML

Design Complexity

Reinforcement learning/deep learning

- We are implementing a reinforcement learning model
- This is when learning is done on a trial and error basis
- Desirable/undesirable outcomes are dictated by a reward function

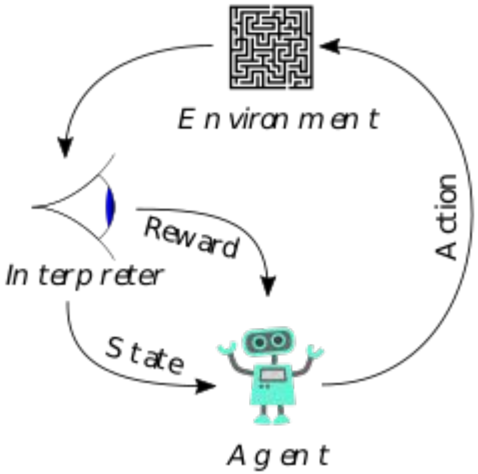


Image courtesy of wikipedia.com

Design Complexity

Network Infrastructure Development

- Maintain services typically provided by AWS for cost efficiency
- S3 Protocol implemented through Open Source software
- Implement SageMaker locally
- RoboMaker has Gazebo as part of ROS

AWS DeepRacer uses the following AWS services to manage required resources:

Amazon Simple Storage Service

To store trained model artifacts in an Amazon S3 bucket.

AWS Lambda

To create and run the reward functions.

AWS CloudFormation

To create training jobs for AWS DeepRacer models.

SageMaker

To train the AWS DeepRacer models.

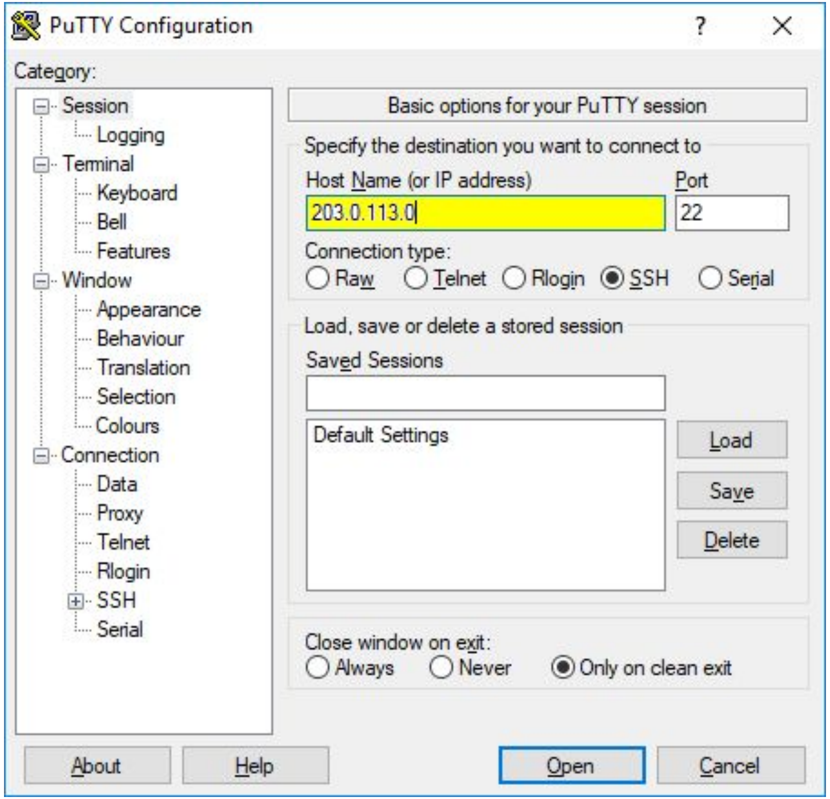
AWS RoboMaker

To simulate an environment for both training and evaluation.

Design Complexity

Desktop Application Development

- Creating a development environment from scratch for students
- Students will create their reward function and manipulate sensors using this application



A Prototype of the DeepRacer

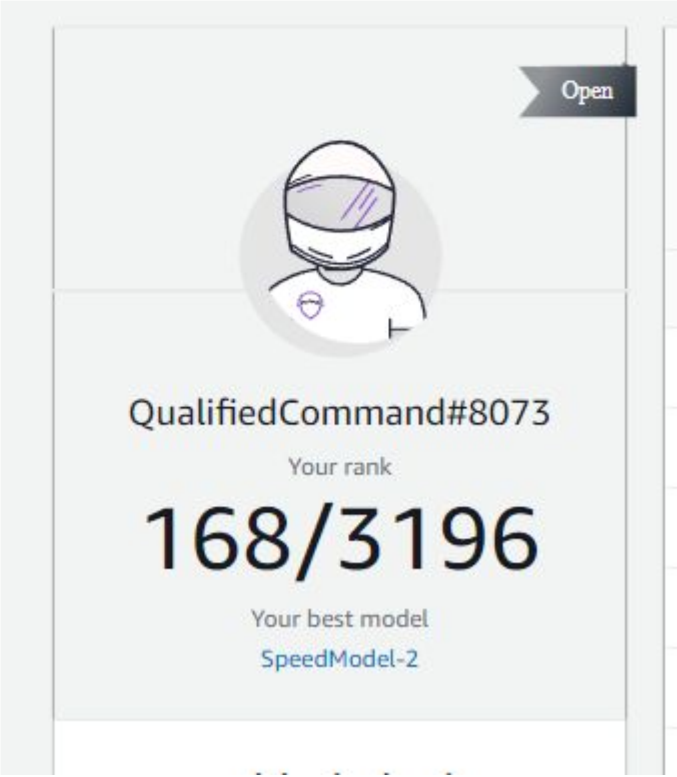
Current implementation

Developed multiple reward functions during our initial exploration of the DeepRacer

Provided exposure to the DeepRacer and its components

Showcased a few issues with implementing such a system/showed issues in being main focal point

We're ranked top 200 in the world for this season!

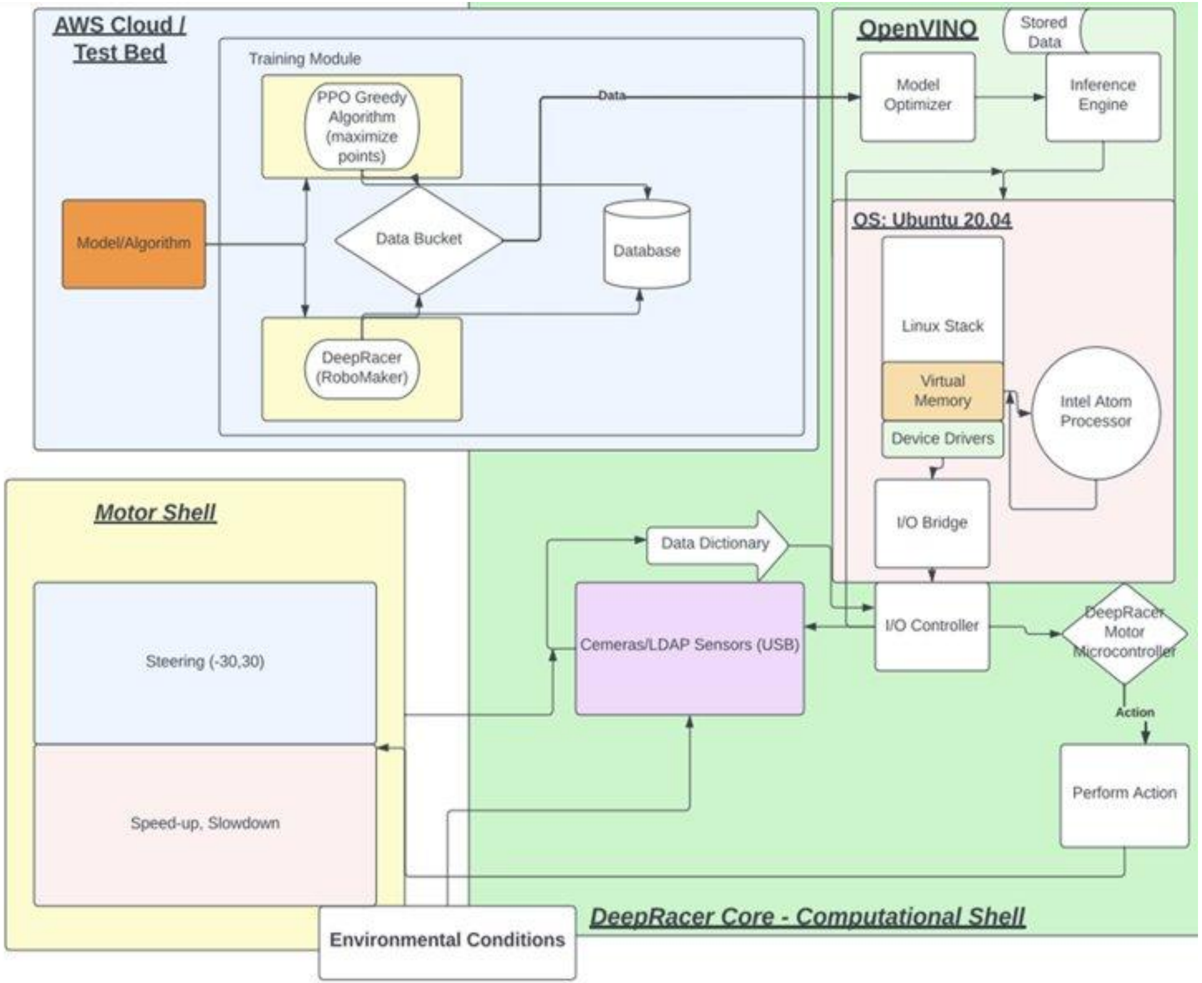


System Design

System Design

Four Components in our Implementation:

- A custom DeepRacer Core
- A custom Testbed (imitated AWS services)
- Physical Hardware/Sensors
- Student focused GUI



The DeepRacer, *an Embedded System*

Images courtesy of aws.amazon.com

The DeepRacer, *a computational system*

<table>
<tr><td colspan="6">NGINX (Web Proxy)</td></tr>
<tr><td>Web Server ROS node</td><td>Web Video ROS node</td><td>Control ROS node</td><td>Battery ROS node</td><td>Media ROS node</td><td>Inference ROS node</td></tr>
<tr><td colspan="6">Root Operation System (Robotic Middleware Application Framework)</td></tr>
<tr><td colspan="6">Ubuntu Operation System (Robot Software Brain Kernel)</td></tr>
<tr><td colspan="6">Compute (Robot Hardware Head)</td></tr>
<tr><td colspan="6">Camera (Robot Hardware Vision Sense)</td></tr>
<tr><td colspan="6">RC CAR (Robot Hardware Body)</td></tr>
</table>

DeepLearning

Introduce Machine Learning via a reward function

Fork of the DeepRacer core, implementing all functionality locally

Interacted with via a desktop application

Built on Embedded System's topics and themes, to the right is configuring the RC Car's MCU

```
void GazeboRosDeepRacerDrive::Load(gazebo::physics::ModelPtr _model, sdf::ElementPtr _sdf)
{
  impl_->model_ = _model;

  auto world = impl_->model_->GetWorld();
  auto physicsEngine = world->Physics();
  physicsEngine->SetParam("friction_model", std::string("cone_model"));

  // Initialize ROS node
  impl_->ros_node_ = gazebo_ros::Node::Get(_sdf);

  // Get QoS profiles
  const gazebo_ros::QoS & qos = impl_->ros_node_->get_qos();

  impl_->max_speed_ = _sdf->Get<double>("max_speed", 4.0).first;
  impl_->max_steer_ = _sdf->Get<double>("max_steer", 0.523599).first;

  impl_->wheel_radius_ = _sdf->Get<double>("wheel_radius", 0.03).first;
  // wheel seperation is the distance between the two rear wheels or two front wheels
  impl_->wheel_separation_ = _sdf->Get<double>("wheel_separation", 0.159202).first;
  // wheel base is the distance between the rear and front wheels
  impl_->wheel_base_ = _sdf->Get<double>("wheel_base", 0.164023).first;

  // Update rate
  auto update_rate = _sdf->Get<double>("update_rate", 100.0).first;
  if (update_rate > 0.0) {
    impl_->update_period_ = 1.0 / update_rate;
  } else {
    impl_->update_period_ = 0.0;
  }
  impl_->last_update_time_ = _model->GetWorld()->SimTime();

  impl_->cmd_vel_sub_ = impl_->ros_node_->create_subscription<geometry_msgs::msg::Twist>(
    "cmd_vel", qos.get_subscription_qos("cmd_vel", rclcpp::QoS(1)),
    std::bind(&GazeboRosDeepRacerDrivePrivate::OnCmdVel, impl_.get(), std::placeholders::_1));


  RCLCPP_INFO(
    impl_->ros_node_->get_logger(), "Subscribed to [%s]", impl_->cmd_vel_sub_->get_topic_name());
```

Implementation Plan for DeepLearning

Project Plan

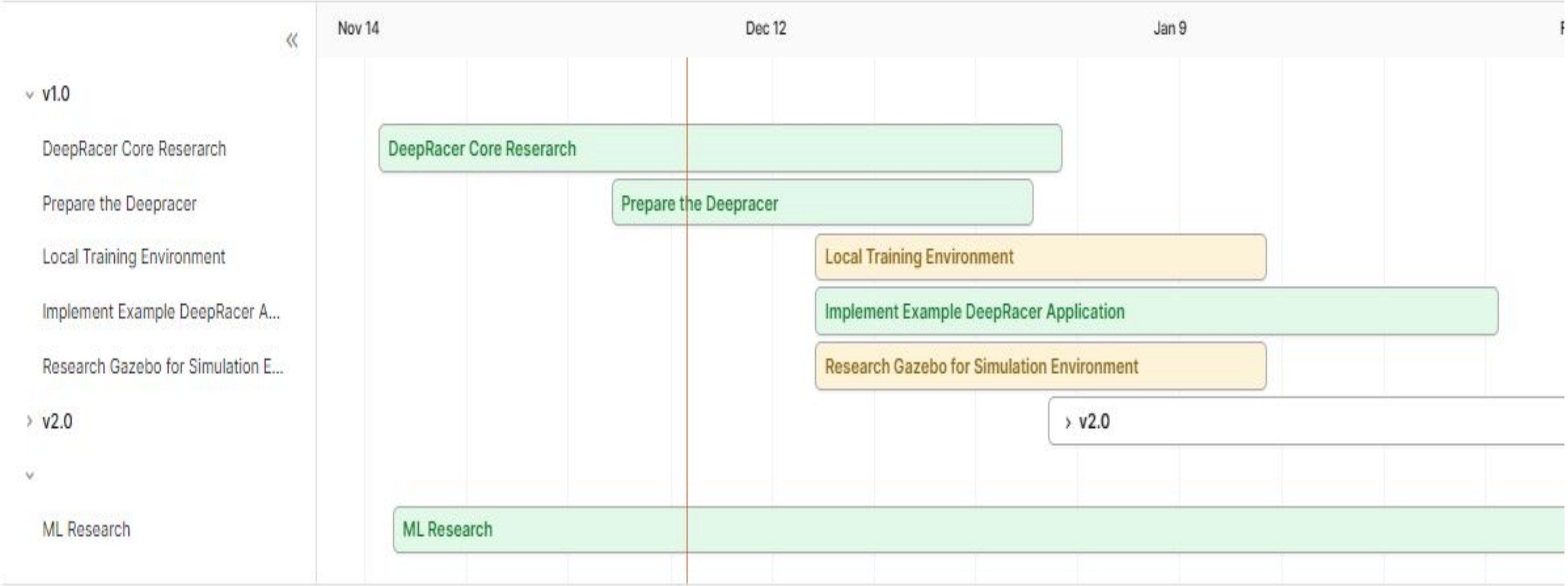
Agile Approach

Phase 1:

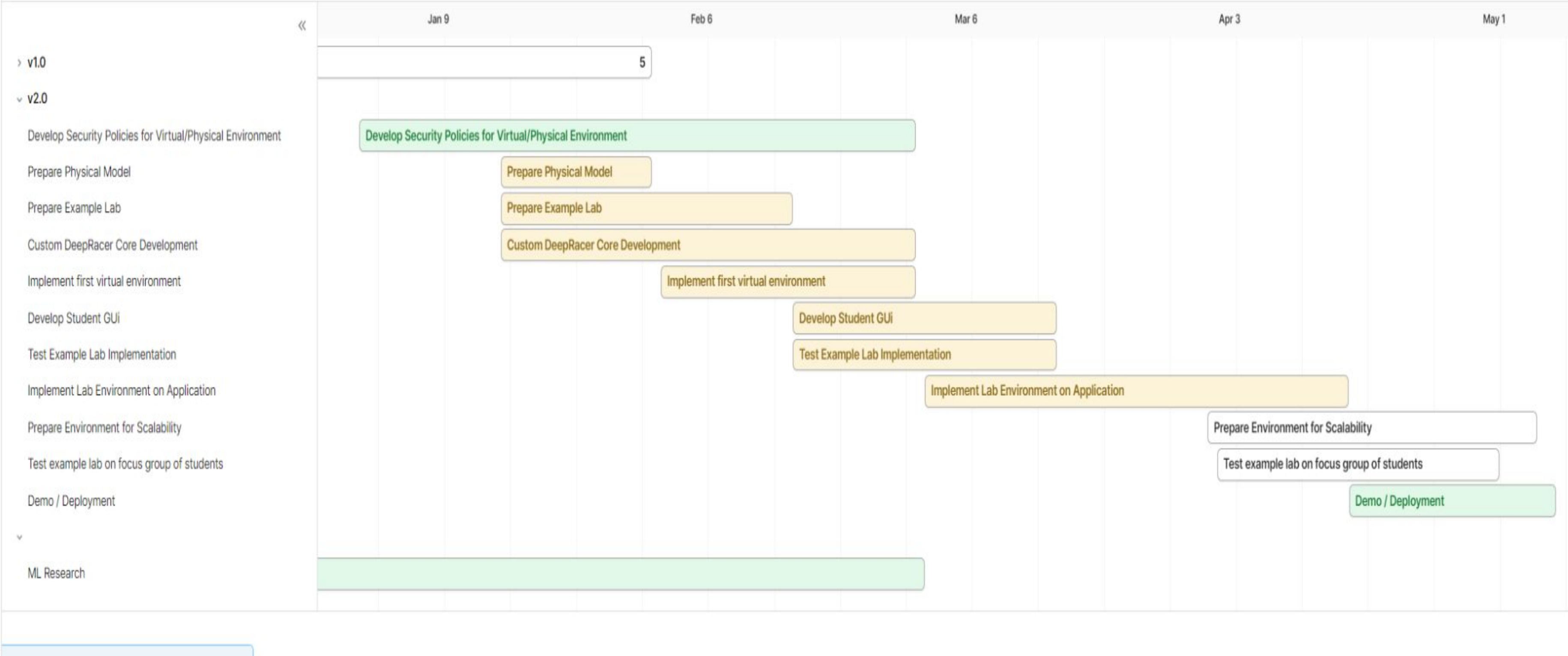
- Complete pivot to new project direction
- Research core technologies we’ll be implementing
- Develop first model ~ January 15th

Phase 2:

- Implement Local Training Module/Hardware
- Develop DeepLearning Desktop Application
- Develop Lab for Focus Group
- Test implementation
- Implement Student GUI



Project Deep Learning: Phase 1 Development



Project Deep Learning: Phase 2 Development

Testing

DeepLearning Application (our DeepRacer Core Implementation):

- Continuously test, gain feedback via focus group, make adjustments to the framework

Educational Materials, and User GUI:

- Experiment with focus group, gain feedback, make adjustments

DeepRacer Testing:

- Unit testing on core nodes
- Implementation testing in virtual environment

Virtual Sandbox Testing:

- Unit, Acceptance, and Regression Testing in comparative analysis vs physical data

Conclusion

Closing

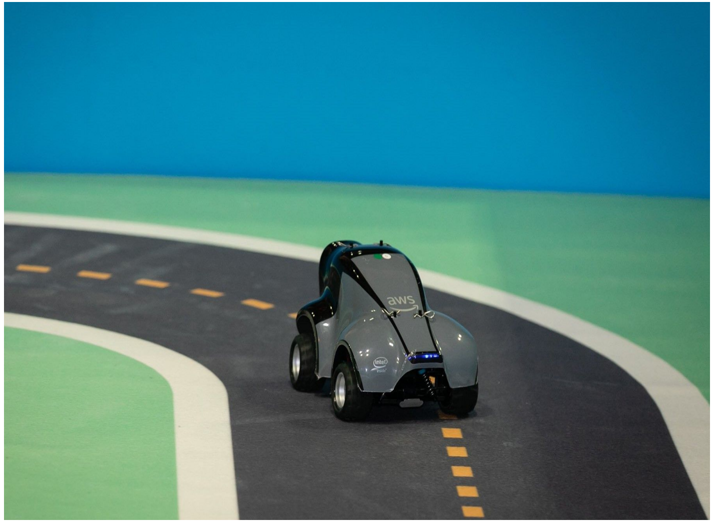
This has been an unusual experience, in a good way

We're taking part of the educational experience just as much as the future students will

Despite the pivot to a new goal, this new track is one we're excited about

For now:

- Wrap up phase one's research and development
- Ramp up experiments with the DeepRacer Core and its components



Thank you!

Sources

AWS DeepRacer_and_Sensor_Guide.pdf (awsstatic.com)