# Towards Robot Navigation with Deep RL

Izzy Brand, Josh Roy, Nate Umbanhowar
CS2951x: Reintegrating AI, Brown University

## The Project

We explored machine learning in an embodied setting in order to investigate the unique challenges and opportunities presented by placing a learning agent in the real world.

We developed a robot, ExplorerBot, capable of navigating the environment and avoiding obstacles using deep reinforcement learning - deep Q-learning, specifically. The neural network ran off-board.

While the robot was unable to learn to navigate from camera data alone, likely due to the unfeasability of collecting a sufficient number of training samples, it successfully learned to drive and avoid obstacles using input from its 4 time-of-flight sensors.

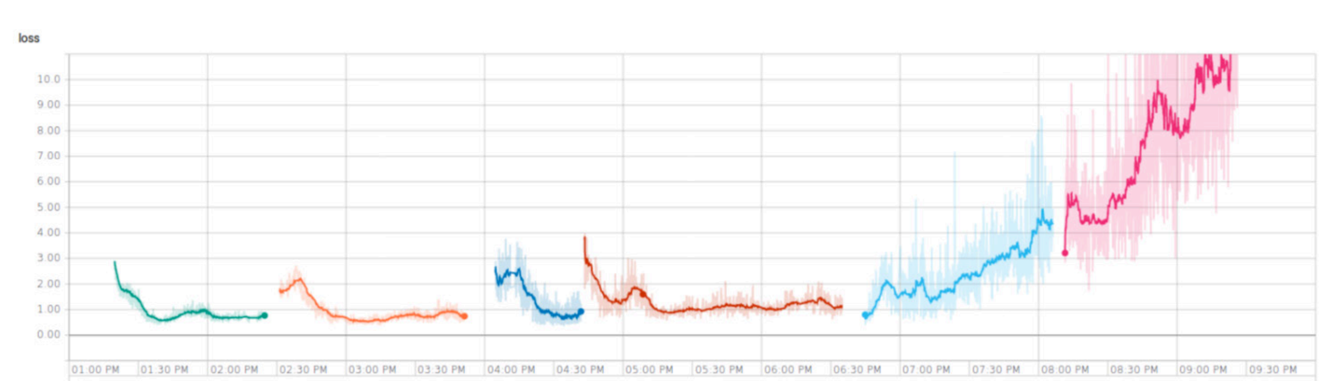
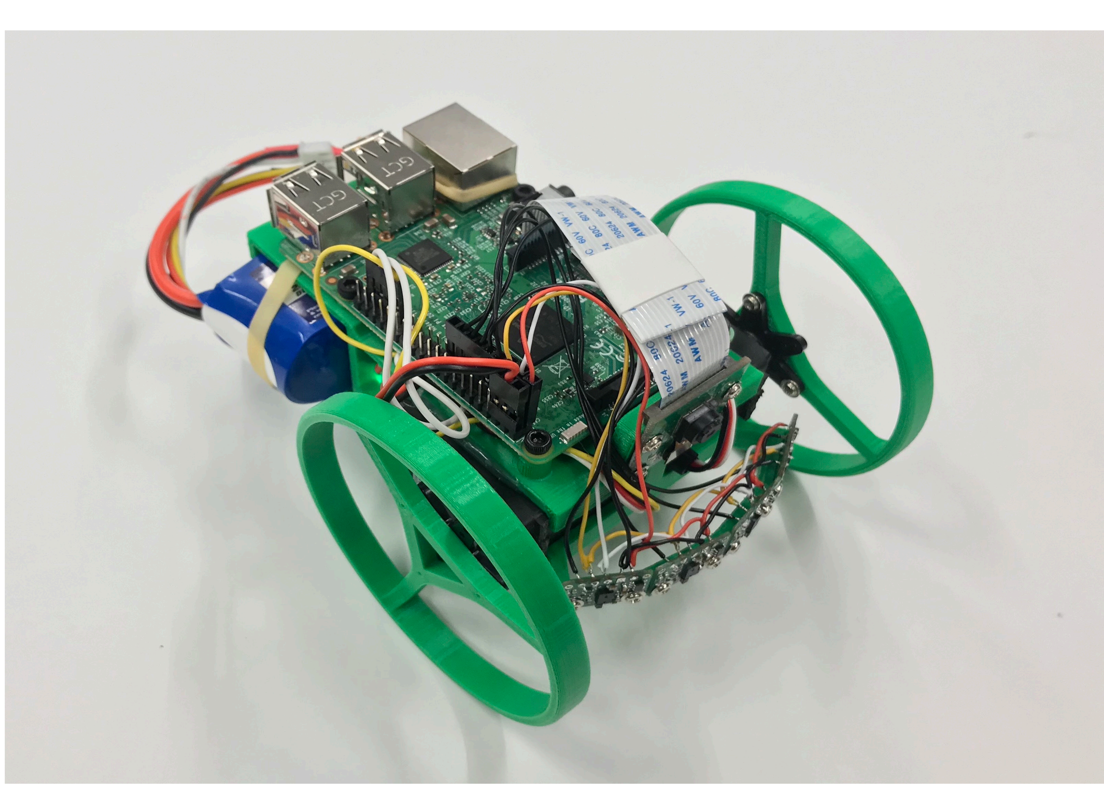
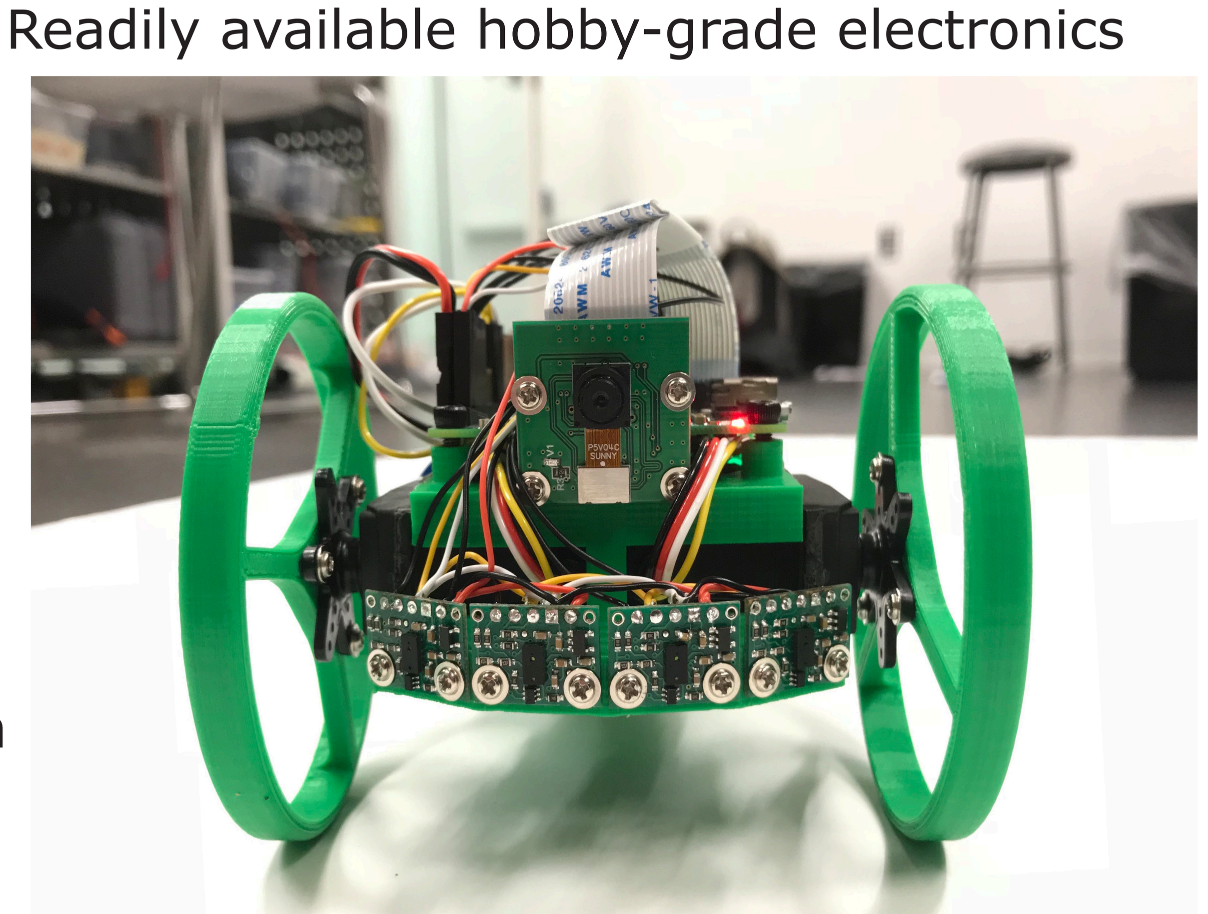


Figure 6: ToF Q-Network loss plotted throughout training. The breaks in the graph are when we changed ExplorerBot's battery. At 6:30pm, around 150,000 iterations, the robot displayed desirable behavior, driving forward and turning to avoid walls. As it kept training, the network diverged and, when tested at 9:30pm in the reward test, the new network performed worse than the network at 6:30pm.

## ExplorerBot



- 3D-printed chassis and wheels
- Total robot cost: $156 USD
- Readily available hobby-grade electronics



- Raspberry Pi 3 based vision system with open source PiCam
- 4 time-of-flight distance sensors

## Deep Q-learning

We model the robot's interaction with its environment as a Markov Decision Process: $(S, A, R, T, \gamma)$

We aim to learn the optimal action-value function

$$Q(s,a) = \mathbb{E}_{s' \sim T(s,a,\cdot)}[r + \gamma \max_{a'} Q(s', a') | s, a]$$

Where Q(s,a) is a neural network. We implement deep Q-learning with experience replay, as described in *Human-level Control Through Deep Reinforcement Learning* (Mnih et. al. 2015) and *Implementing the Deep Q-Network* (Roderick et. al. 2017).

**Algorithm 1** Deep Q-learning with experience replay [9, 12]

Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
**for** episode $1, M$ **do** Initialize sequence $\bar{s}_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(\bar{s}_1)$
  **for** $t = 1, T$ **do**
    With probability $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \arg\max_a Q(\phi(\bar{s}_t), a; \theta)$
    Execute action $a_t$ in the emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $\bar{s}_{t+1} = \bar{s}_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(\bar{s}_{t+1})$
    Store experience $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
    Sample random minibatch of experiences $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the weights $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  **end for**
**end for**

## The Future

- Work to increase loop frequency, to enable faster convergence of DQN and better results on training from camera data.
- Use transfer learning to speed the learning of image features from camera data.