Balancing Bot - PID versus Reinforcement Learning

- Chenning Yu, chy010@eng.ucsd.edu, A53305427 (solo project)

Motivation

Currently, the RL algorithm for inverted pendulum problem is quite mature, which is exactly equivalent to controlling a balance bot. However, traditionally these algorithms are only trained in a simulated environment, and such environment lacks the ability of dealing with physical reality, which has signals with noise.

On the other side, although there are already a dozen of demos on the web showing how to build a Segway based on Arduino or Raspberry Pi, these demos are only played in a fixed landscape, which contains constant friction coefficient. As a result, the control algorithm might suffer from generalizing to other environments, which has a totally different friction constant. An RL-based learning algorithm can solve this issue, because it can learn new parameters from trail-and-errors in new environments.

As a result, I would love to compare the PID algorithm with the reinforcement learning techniques. The observation might also help explain why currently there is no mature deployment of reinforcement learning algorithm in closed-loop control algorithms.

Related Work

The original Q-learning paper [1] comes around 1995, where it introduces the Q function in order to decouple the system transition function from the policy in the value evaluation scenario. Later around 2013, people starts using neural network as a approximation heuristic to evaluate the Q- value in more complex environments. This approach starts the trend of using neural networks in reinforcement learning areas.

With implementing a Q-learning network, Rahman et al.[3] applies reinforcement learning on the self-balancing bot problem, within a simulated environment. Chang et al.[4] uses a Arduino self-balancing car as the platform to conduct reinforcement learning. However, this approach deviates from the inverted pendulum problem, since the center of mass does not lie relatively far above the motors.

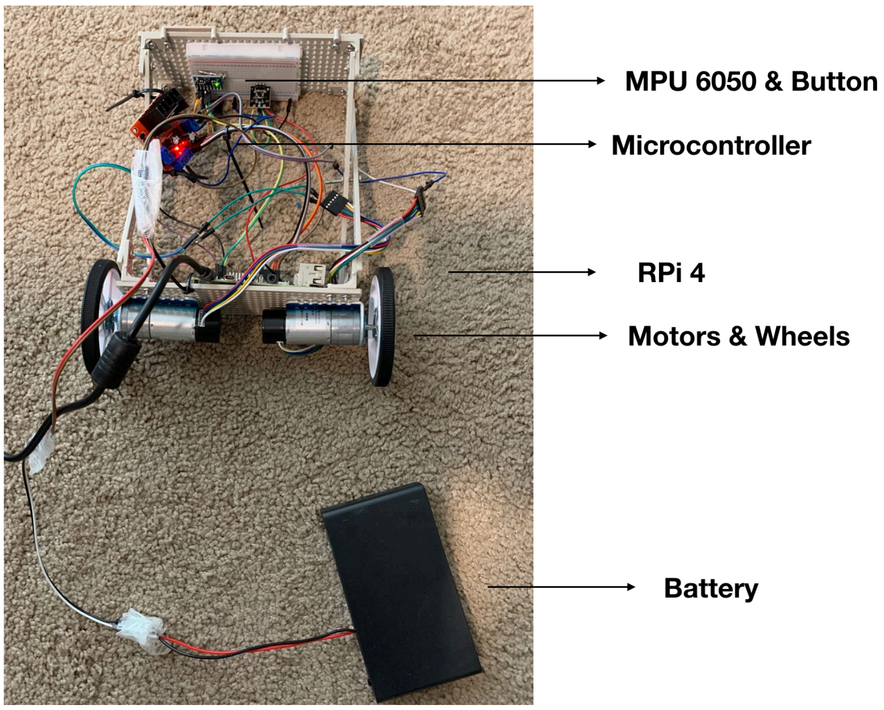
My approach's uniqueness: To my knowledge, this is the first experiment on the balancing bot, with a focus on fair comparisons between PID control algorithm and reinforcement learning algorithm. Furthermore, the car frame is also manually built, which is adjustable according to experimental demands.

Hardware Components

In particular, the hardware components include:

- Raspberry Pi 4
- Two motors with 12V 600 rpm
- L298N motor drive controller
- Two motor gear brackets
- Battery cover pack
- Two multi-hub wheels
- Universal arm set and Universal plate set

I choose motors with 12V, simply for the convenience to connect the motor with L298N. Furthermore, it has encoders to count the rounds, which might be useful to enrich the information of the states of the balancing bot.



It might be better to choose a motor with less rpm, since less rpm means more torque, which would be helpful for driving the motor against the friction from coarse grounds.

I choose the universal arm set and plate set, since it is easy to mount the chips onto the car using self locking wire ties, and the arm itself can be broken into several pieces if needed.

Software Components

I use python as our main programming language, since it works consistently with several popular deep learning frameworks. I choose PyTorch as our deep learning framework, since it is easy to check out some intermediate results. The python version is 3.6.0. I compile PyTorch locally using version 1.2.0, with the numpy version 1.15. For hardware libraries, the GPIO version number is 0.7.0, and the smbus version number is 1.1.post2.

For the control algorithms, I implement PID and Q-learning with the following details:

Software Components - Control Algorithms: PID

PID algorithm takes the angle and angular acceleration as the inputs, and it estimates the smoothed angle over time using complementary filter with alpha=0.02.

Using the error between the smoothed angle and the ideal angle, the program calculates the P-term and D-term using Kp=5 and Kd=12.8565. The I-term is ignored, because it does not produce better results in our experiment. After summing up the PID term, the motor changes its speed with interfaces of GPIO. The ideal angle is evaluated as the angle in the initial stage. As a result, the balancing bot is needed to keep balanced when it is activated to running.

Software Components - Control Algorithms: Q-learning

The policy is defined as choosing between 21 discrete actions between [-100, 100], where the value means is the PWM speed. The Q-network shares parameters with policy network, and the total network is described as follows:

The network's input is the same: the current angle and angular acceleration. There are 10 hidden neurons in the hidden layer, and it is followed by two output heads: (1) the policy head, which is a head of 21 outputs using linear layer followed by the softmax operation; (2) the value head, which has one output using a linear layer.

All the training and inferring is deployed locally on raspberry pi. We sample one trajectory each time. After the bot falls off, the program feeds the sample to the neural network to refresh its own weight.

To reduce testing time, we integrate a button to the breadboard. When the button is pushed, the bot starts running. The program automatically pauses running when it detects its angle is above some threshold, in particular, 70 degrees.

Software Components - Control Algorithms: General Looping

Both above algorithms have a precise demand on the looping time. As a result, I explicitly set the control interval as 35ms, and when the calculation time costs less than 35 ms, the controlling algorithm continues to sleep until 35ms is spent, and it goes to the next loop.

Hardware & Software Integration

Hardware & Software Integration - Motor Torque

One challenge I faced is that the motor torque is a little low. As a result, if the car chassis is heavy, the bot cannot really balance itself because it is even not feasible to move against the friction.

As a result, I put the battery out of the bot, which makes the bot's weight much lighter.

Hardware & Software Integration - Angle Smoothing

One intuitive thing to do for getting angles is to directly get the angle from the accelerator. However, it often does not work, when the motor is trembling, and the accelerator's value is not stable. As a result, it is more helpful to estimate the angle using Baysian filter such as complementary filter, which is a method to calculate the angle smoothly over time using an interpolation between the estimation from gyroscope and the estimation from accelerator.

Experimental Section

In the experiment, the goal is simple: to make the bot stand vertically as long as possible. In particular, stand vertically means the angle is within the threshold (-70, 70). Once the angle falls out of this region, we stop timing.

Experimental Section - Results

The result is counter-intuitive: **the RL methods cannot really make the bot keep balanced, and its average time is below one second. The PID algorithm, on the other side, takes average at the upper bound of timing.** The performances of two methods has a huge gap: one is close to zero, and the other is close to the infinity.

Experimental Section - Implications

Though the result seems to be counter-intuitive, there are still some explanations on it:

1. The reward design. The design needs well-designed for reality. On the other hand, the reward is explicitly given in simulated environments, which are good for environments such as Atari game and Poker, Go.
2. The model and sampling design. It might be better to have bigger models and more data.
3. The real world knowledge. This is the most important thing. PID utilizes the knowledge as a belief that there must a balancing point, where RL does not have such knowledge.

Conclusion

In summary, I design a balancing bot, from hardware to software, starting from zero. I deployed PID control algorithm on the balancing bot successfully, which is a key concept in the class.

If I have more time and money, I would like to buy a pair of motors with higher torque, and try out other reinforcement learning algorithms and find out the results, since it might not be the case that RL is not approachable, but it is just that my implementation and methodology deviates from the correct one.

References

[1] Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[3] Rahman, M. M., Rashid, S. H., & Hossain, M. M. (2018). Implementation of Q learning and deep Q network for controlling a self balancing robot model. Robotics and biomimetics, 5(1), 8.

[4] Chang, C. L., & Chang, S. Y. (2016, December). Using reinforcement learning to achieve two wheeled self balancing control. In 2016 International Computer Symposium (ICS) (pp. 104-107). IEEE.