Inverted Pendulum on a Quadcopter: A Reinforcement Learning Approach

Physical Sciences

Alexandre El Assad aelassad@stanford.edu
Elise Fournier-Bidoz efb@stanford.edu
Pierre Lachevre lpierre@stanford.edu
Javier Sagastuy jvrsgsty@stanford.edu

December 15th, 2017

CS229 - Final Report

1 Motivation

Current quadcopter stabilization is done using classical PID controllers. They usually perform well expect for:

- altitude control, due to complex airflow interactions present in the system.
- when non-linearities are introduced, which is the case in clustered environments.

We envision reinforcement learning as a major breakthrough in the field of robotic control as it eliminates the need for a predefined controller structure which limits the overall performance and costs more human effort. In other words, reinforcement learning avoids creating manually-tuned control algorithms to complete specific tasks. This technique could lead to building tailored, adaptable, autonomously learned controllers for drones and other types of robotic technologies.

2 Problem definition

Our project consists primarily of three parts: a physics simulator of the dynamics of the coupled quadcopter and inverted pendulum system, a reinforcement learning algorithm, and a reward function that the RL algorithm will attempt to maximize.

Schematically, the problem can be illustrated as follows:

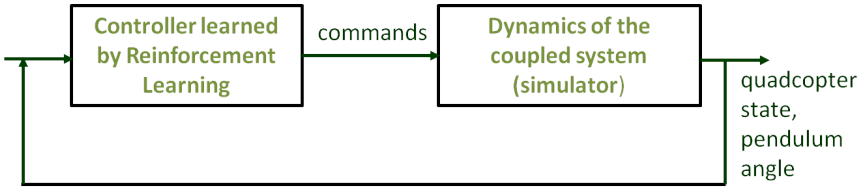


Figure 1: Illustration of the controller / system interaction

Once the learning is accomplished, the controller - learnt by RL - outputs the optimal action to take at each time step, leading to a maximization of the expected value and hopefully a longer stabilization of the pendulum.

3 Modeling and solving approach

3.1 Simplified dynamic model

The Matlab simulator is based on the dynamics outlined in the paper by Hehn et.al. [1]. These dynamics couple the dynamics of a quadcopter with an inverted pendulum attached to its center of mass. However, in the simplified dynamic model, we eliminated the Euler angles and kept only translational quantities. This is equivalent to considering the drone as a platform with three degrees of mobility x, y and z. The state vector X is $[x, y, z, \dot{x}, \dot{y}, \dot{z}, r, s, \dot{r}, \dot{s}] \in \mathbb{R}^{10}$. $[x, y, z, \dot{x}, \dot{y}, \dot{z}]$ stand for the position and velocity of the drone, whereas $[r, s, \dot{r}, \dot{s}]$ represent the position and velocity of the center of mass of the stick with respect to the drone's center of mass, i.e. the displacement of the pendulum's center of mass with respect to the drone. The control vector U is $[a_x, a_y, a_z] \in \mathbb{R}^3$, the x, y, z accelerations of the drone's center of mass. We discretized the action space to a simple 5^3 space where x, y could take values in $\{-2, -1, 0, 1, 2\}$ and z could take values in $\{0.5, 0.75, 1, 1.25, 1.5\} * 9.81$. From this model, a discrete time update function was developed.

3.2 Full dynamic model

The full dynamics include the orientation of the quadcopter. The state vector differs from the previous one in that it includes Euler angles (i.e. roll, pitch and yaw). The state vector X is $[x, y, z, \dot{x}, \dot{y}, \dot{z}, \alpha, \beta, \gamma, r, s, \dot{r}, \dot{s}] \in \mathbb{R}^{13}$ (position and velocity of the drone, Euler angles, position and velocity of the center of mass of the stick). The control vector U is $[a, \omega_x, \omega_y, \omega_z] \in \mathbb{R}^4$ and represents the mass normalized collective thrust and the rotational rates about the vehicle body axis, which standard for control of some drones. We used the same discretized action space as with the simplified dynamic model. From this model, a discrete time update function was developed.

3.3 Solving strategy : Reinforcement Learning

We chose to implement fitted value iteration for two reasons: first because we're dealing with continuous variables, second because the state space is too large to be discretized and solve the MDP directly (depending on which dynamic model is chosen, it is either 10 or 13 dimensional). This models the value function as $V(s) = \theta^\intercal \Phi(s)$.

Several Matlab files and functions were used to decompose the value iteration algorithm into blocks. The implementation includes: `FVI.m` where we coded fitted value iteration from the course notes (Chapter 13 on Reinforcement Learning and Control), `feature_map.m` which contains our feature vector, `reward.m` which contains our reward function, `getSuccessor.m` which contains the dynamic model, `gradient_descent.m` which takes care of finding θ that minimizes the cost function (either by gradient descent, closed form solution or using `regress` command), `ind2action.m` which discretizes the action space, and `control.m` which runs a simulation once the learning is done and a weight vector fixed. We also implemented a 3D animation for the poster session, to show real-time stabilization using the full dynamic model. This animation shows how the quadcopter moves (body rotations followed by thrust) in order for the stick to stabilize.

3.4 Feature mapping and reward design

Once the solving strategy was set, the next step was to design an appropriate feature mapping as well as a reward model. After some trial and error as well as analysis of the geometry of the problem, we reached the following conclusions:

- Since our problem is symmetric in all coordinates with regards to the origin, all components of the mapping should be symmetric. We therefore elected to remove all linear terms from it. Furthermore, the geometry is also symmetric with regards to rotation in the (x, y) plane, and thus coefficients for x and y components of position and velocity of pole and drone should be the same. We however elected to leave those as separate coefficients and checked if they were equal after learning. This also would allow us to include non symmetric perturbation (such as wind) to the learning process. Concretely, we have defined $\Phi(s) = [1, x^2, y^2, z^2, x\dot{x}, y\dot{y}, z\dot{z}, \dot{x}^2, \dot{y}^2, \dot{z}^2, r^2, s^2, r\dot{r}, s\dot{s}, \dot{r}^2, \dot{s}^2] \in \mathbb{R}^{16}$.
- After experimenting with many reward functions (some of the continuous), it appeared that simpler ones allowed for better convergence, which is why we settled on a binary one. Concretely, we have defined

$$r(s) = \begin{cases} -1 \text{ if } |x| > 5 \text{ or } |y| > 5 \text{ or } |z| > 0.5 \text{ or } |r| > 0.3 \text{ or } |s| > 0.3 \\ 0.01 \text{ otherwise} \end{cases}$$

4 Results and analysis

We generated the dashboard shown in Fig. 2 to better visualize the state evolution of our drone and evaluate our results. We only report a subset of a results due to the length constraint on the project report.

4.1 Results for the simplified dynamic model

- Without perturbations :
  After running Fitted Value Iteration to obtain the optimal value function, we tested the logic on the simplified model (i.e. the actions are modeled as the acceleration of the flying platform). Results are shown in Fig. 3a. We can see that the pole reaches a stable position in a very short time (less than 2.5s) while the drone takes longer to come back to the origin. This is due to the very high weights on the position of the pole in the feature mapping $\Phi(s)$.
- With perturbations :
  To test the robustness of our value function, we then added external forces on the drone such as wind (normally distributed acceleration with a non-zero mean). The results are shown in Fig. 3b. We can see that the

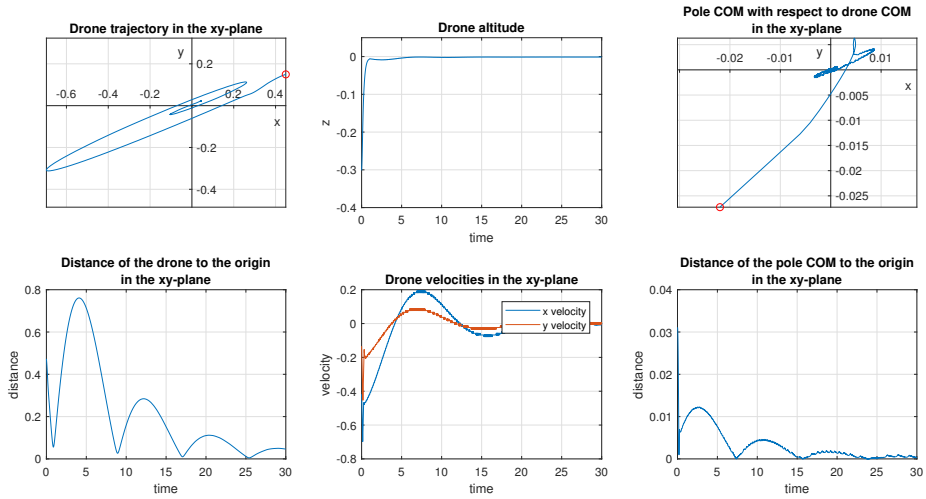
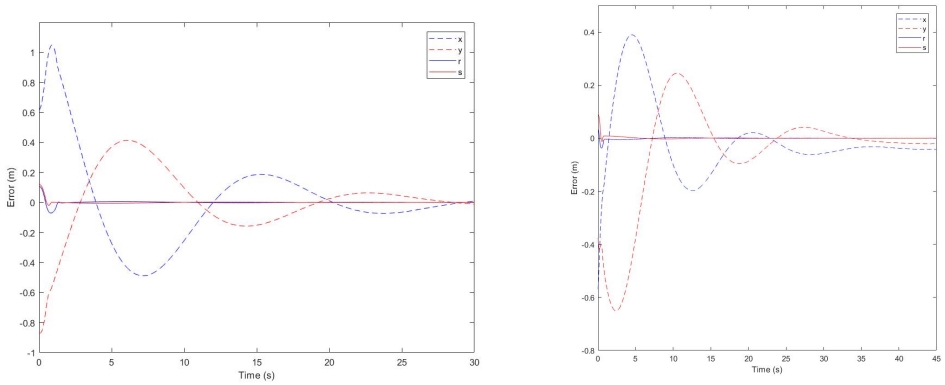


Figure 2: State visualization dashboard for a run on the simplified model without perturbations



(a) Pendulum position error (r, s) and quadrotor position error (x, y) without perturbations

(b) Pendulum position error (r, s) and quadrotor position error (x, y) with perturbations

Figure 3: State errors during balancing

perturbation does not affect the result (both plots are very similar). It was quite a surprise to see our algorithm handle this well external forces without having to retrain to get a new set of weights.

4.2 Results for the full dynamic model

Once the learning is accomplished **on the simplified model**, the controller -learnt by RL- outputs the optimal action to take at each time step, in terms of accelerations a_x, a_y, a_z. These actions are fed into a an action converter that transforms them into the realistic commands: a (the thrust), and ω_x, ω_y, ω_z (the body rotation rates). These last four quantities are what a RC pilot would actually control. The action converter takes into account the **delay** between a, ω_x, ω_y, ω_z and the resulting a_x, a_y, a_z. Indeed, when commanding thrust and rotation rates, the drone first rotates by a certain amount before translating. Therefore commanding a, ω_x, ω_y, ω_z doesn't result instantaneously into a_x, a_y, a_z, which needs to be accounted for.

The strategy can be summarized by the diagram shown in Fig. 4.

Our model turned out to be very robust even with those added dynamic effects. It was in fact able to handle the same perturbations as the ideal model. In Fig. 5 we show some of the results from our dashboard both with and without perturbations.

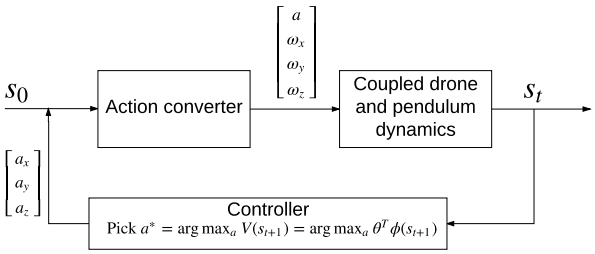
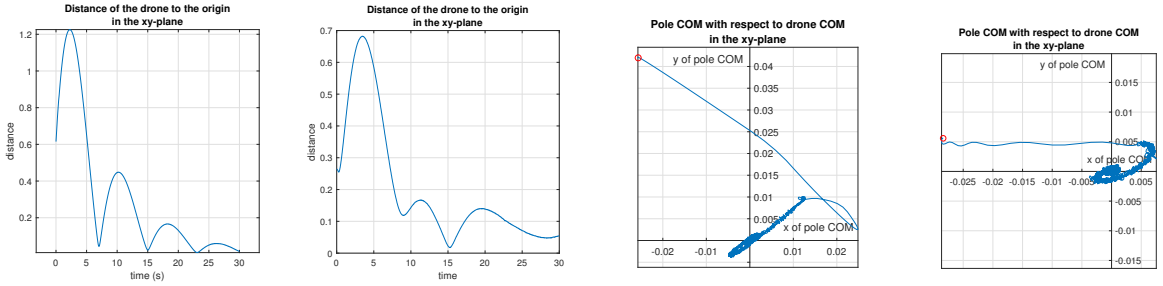


Figure 4: Illustration of the controller / system interaction when modeled with full dynamics



(a) Distance to origin without perturbations

(b) Distance to origin with perturbations

(c) Pendulum center of mass displacement without perturbations

(d) Pendulum center of mass displacement with perturbations

Figure 5: Full model performance with and without perturbations

To further study the robustness of our model, we can limit the angular rates that our converter is outputting. Although the values of angular velocity we obtain without saturation are perfectly achievable by research grade drones, we wanted to assess whether consumer drones would perform well. We have thus saturated the angular rates at $300^o/s$, typical in the industry, and ran our RL controller. In the end it does manage to balance the pole (the error is bounded), but it is unable to bring it to 0. The results are quite promising, and could probably be improved with additional learning and an extended mapping function.

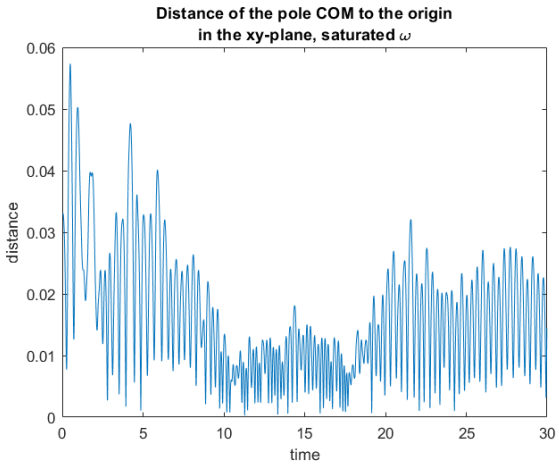


Figure 6: Pendulum center of mass error with angular rate saturation

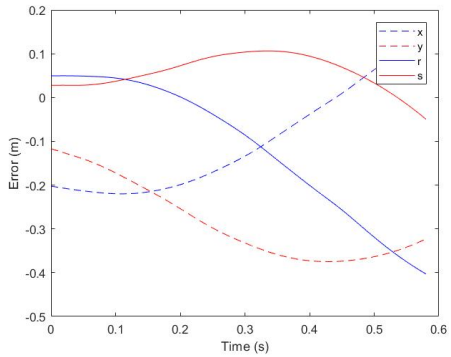


Figure 7: State errors during balancing: Pendulum position error (r, s) and quadrotor position error (x, y) with noisy observation of the state

4.3 Error analysis

Most of the error analysis was performed for the milestone. This led us to implement and run Fitted Value Iteration on the simplified dynamics model to give us a more intuitive simulator and make it easier to debug and optimize the algorithm. The two main takeaways from the error analysis of the milestone were:

- Analyzing the coefficients obtained from the learning is essential to improving our results. For example, correct signs can be deduced from our intuition of the system, but the learning might converge towards an incorrect value (another local optimum). Therefore, initializing our coefficients with the correct signs had a great impact on our performance.

- The reward needs to be adequate to the expected behavior. In one of our early prototypes, we had weighted position of the drone far less than the balance of the pendulum. Value iteration thus converged to a policy causing the drone to accelerate downwards, which does balance the pendulum thanks to inertial forces. While this is an extreme case of reward hacking, other less bizarre scenarios occur quite often, such as the system converging to a non zero value (thus minimizing velocity and pendulum balance but not drone position). The main way to correct this has been to weigh all factors equally. One strategy that has been done in literature (see Figueroa et.al. [2]) is to train two different controllers: one that only prioritizes balance used when the pendulum is at risk of falling, and another one that emphasizes a balance-maintaining policy once the pendulum is reasonably balanced.

5 Challenges

Selecting an appropriate feature mapping and reward function were two of the biggest challenges we encountered. Also, the computational time required for the action converter shown in Fig. 4 to change the action $[a_x, a_y, a_z]$ output by the RL controller into the action $[a, \omega_x, \omega_y, \omega_z]$ accepted by the drone was taking very long since it involved solving a very constrained system of equations. We solved this by finding a closed form solution analytically.

6 Future work

In the current version of the simulator, sensors are assumed to be perfect. However this is never the case in a real life environment and this is why we tried adding some noise on the state observation. Results are shown in Fig. 7. In all the cases, the drone could not stabilize the pendulum. Future work would include re-training Fitted Value Iteration to have a more robust policy.

Furthermore, we would like to implement our learned policy on a real drone. If we handle the case of noisy observations, and because the current policy can handle external perturbations, this could lead to very stable drones in clustered environment or in the case of ground effect (when airflow "hits" the ground and is re-ingested by the propellers leading to perturbations that cannot be simulated).

7 Contributions

Overall, we believe all team members contributed equally with work and ideas to the project. A more precise breakup of what everybody worked on follows.

- Elise: implemented the full dynamics simulator in Matlab, debugged value-iteration and helped with training on the simple and full dynamic model, participated in the animation implementation.
- Alexandre: simulator implementation in Matlab (simplified dynamics), fitted value iteration for the simplified dynamic model with various sets of features, participated in the animation implementation, worked on converter for full dynamic model, cleaned the code to get its final version.
- Pierre: implemented several versions of fitted value iteration (both simplified and full dynamics), assessed them on a number of mappings and reward functions, worked on converter for full dynamic model.
- Javier: implemented and debugged several versions of fitted value iteration and feature mappings. Tried an implementation of gradient descent to minimize the cost function. Tried different initialization techniques. Designed and wrote the poster.

References

[1] Markus Hehn and Raffaello D'Andrea. A flying inverted pendulum. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 763–770. IEEE, 2011.

[2] Rafael Figueroa, Aleksandra Faust, Patricio Cruz, Lydia Tapia, and Rafael Fierro. Reinforcement learning for balancing a flying inverted pendulum. In *Intelligent Control and Automation (WCICA), 2014 11th World Congress on*, pages 1787–1793. IEEE, 2014.

[3] Samir Bouabdallah and Roland Siegwart. Full control of a quadrotor. In *Intelligent robots and systems, 2007. IROS 2007. IEEE/RSJ international conference on*, pages 153–158. Ieee, 2007.

[4] Kangbeom Cheon, Jaehoon Kim, Moussa Hamadache, and Dongik Lee. On replacing pid controller with deep learning controller for dc motor system. *Journal of Automation and Control Engineering Vol*, 3(6), 2015.

[5] Angela P Schoellig, Clemens Wiltsche, and Raffaello D'Andrea. Feed-forward parameter identification for precise periodic quadrocopter motions. In *American Control Conference (ACC), 2012*, pages 4313–4318. IEEE, 2012.

[6] Markus Hehn and Raffaello D'Andrea. Quadrocopter trajectory generation and control. *IFAC Proceedings Volumes*, 44(1):1485–1491, 2011.

[7] Sergei Lupashin, Angela Schöllig, Michael Sherback, and Raffaello D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1642–1648. IEEE, 2010.

[8] M Sedighizadeh and A Rezazadeh. Adaptive pid controller based on reinforcement learning for wind turbine control. In *Proceedings of world academy of science, engineering and technology*, volume 27, pages 257–262, 2008.

[9] Xue-Song Wang, Yu-Hu Cheng, and Sun Wei. A proposal of adaptive pid controller based on reinforcement learning. *Journal of China University of Mining and Technology*, 17(1):40–44, 2007.

[10] Shin Wakitani and Toru Yamamoto. Design and application of a data-driven pid controller. In *Control Applications (CCA), 2014 IEEE Conference on*, pages 1443–1448. IEEE, 2014.