Machine Learning Engineer Nanodegree

Capstone Project

Omar Castro
August 18, 2018

I. Definition

Project Overview

Robot grasping is one of the areas that is attracting more attention both in Machine Learning and in Robotics nowadays. However, current Reinforcement Learning (RL) algorithms typically require thousands or trials to learn controllers. This is impractical in robots because repetitions are expensive, time consuming, and robot parts suffer from tear and wear from repeated use. Another solution is to use simulators, but robots come in countless different forms, measures and dynamics, so what is learned in a simulator may not by applicable to a real-world robot.

This project aims at using a data efficient and replicable process to teach a low cost 6 DOF robot arm to reach any position on a plane to grasp and manipulate objects. The main challenges of the project were to find the functions with as little observations as possible, and achieve this with low cost robot arms, which are noisy machines and are not built to last. One of the most cited papers in data efficient robot learning is [1], where Deisenroth and Rasmussen teach a low-cost robot manipulator to perform a basic task with very few trials using PILCO (probabilistic inference for learning control), a data-efficient model-based policy search method [2]. However, PILCO requires large computational times to optimize the policy, since it relies on computationally expensive methods [4], [7].

A more promising approach to achieve frugal robot learning seems to be Bayesian Optimization (BO). Calandra used BO to teach a low cost bipedal to walk with a steady gait [4]. BO is a state-of-the-art model-based approach to optimization under uncertainty. The solution proposed in this project is to estimate a series of BO models to approximate the functions relating the coordinates in the plane with the angles of the servos. A simulator was created using a GPR to estimate the function relating coordinates with angles for each

DOF of the robot. Additionally, the model was successfully replicated on a second low-cost manipulator with different characteristics and with similar results.

Problem Statement

The problem this project tries to solve is how to teach any 6 DOF manipulator to reach any point on a plane given its coordinates. We assume that there is no expert knowledge of the robot, and that there is no specific simulator available to model it. To reach a point on a plane a controller of the robot should send the desired angle to each of the 6 servos of the arm. To solve the problem, the following steps were taken:

1. Perform a series of trials with the robot arm, setting different angles to the robot servos to reach several points on a plane changing the Y coordinate but keeping the X coordinate fixed. With this step the angles of the 3 vertical servos (see figure 2) could be related to the distance of the point to the base of the arm. The result of this step was a database with distances from the base of the arm to the points on the plane and with the angles of the 3 vertical servos corresponding to each distance.

2. Perform a series of trials with the robot arm setting different angles to the rotation servo of the robot (see figure 2) to reach several points in the plane but maintaining fixed the distance from the base of the robot to the points. This second step was necessary to relate the angles of the servo that controls the rotation of the arm with the angle of the points in the plane.

3. Create a series with the actual angles of the points reached in step 2. To create this group of angles the distance from the base of the arm to the point in the plane was calculated using the Pythagorean formula, and the angle of the point was calculated using a trigonometric formula.

4. Estimate the functions relating the distances of the points to the base of the arm with the angles of the 3 vertical servos, and the function relating the actual angle of the point with the angle of the rotation servo.

5. Create a simulator with the 4 functions to predict the combination of the angles of the 4 servos given the coordinates of any point in a plane.

6. Test the results of the simulator using the real robot arm to reach a series of random points on a plane.

7. Estimate the functions relating the coordinates of the points and the angles of the servos using 2 other different models as benchmark.

8. Reproduce all the previous steps with a different low-cost 6 DOF arm to test the reproducibility of the model.

Metrics

The only metric used in this project is the success rate of the robotic arm to reach the specified point in the plane using the angles predicted by the simulator. To make the exercise more real, the arm had to grasp a one cubic centimeter cube positioned on the points to be reached. The success rate was measured by the number of successful trials grabbing a cube divided by the total number of trials. Since the manipulators used in this project were low-cost, it was important to minimize the number of trials, because the parts of the arms and the servos are not designed for heavy work, and they suffer from tear and wear very quickly.

II. Analysis

Data Exploration

The manipulator used for the project was a 6 DOF Robot Arm LewanSoul LeArm that was bought on Amazon for US$ 129.99. All the data for the project comes from the angles of the 6 servos of the arm and the position of the tip of the claw on a plane for each combination of angles.

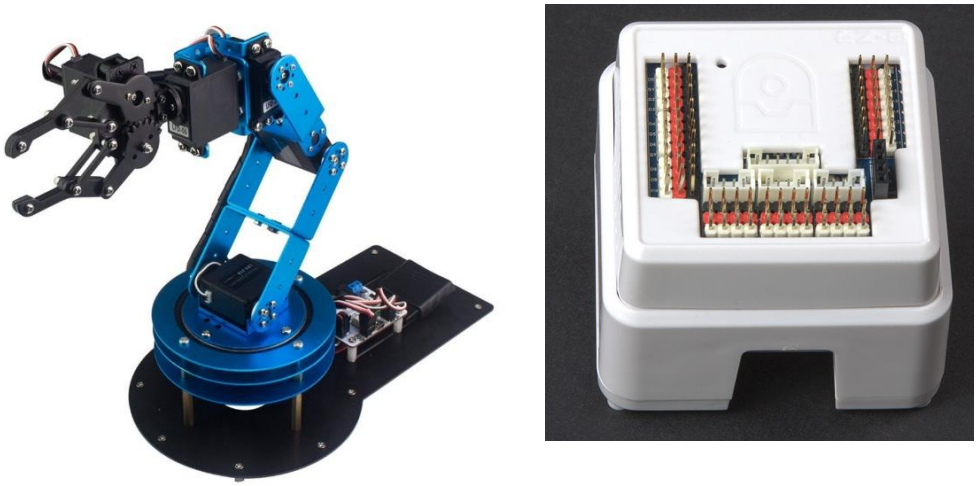

Figure 1: 6 DOF Robot Arm LewanSoul LeArm and Robot controller EZB-V4

The robot controller EZB-V4 was used to manipulate the arm. This controller powers more than 20, 000 robots worldwide, according to their creators [5]. The controller costs US$ 79.99 on the web page of ezrobot. The software that handles EZB-V4, Ez-Builder [6], is a free application provided by the same company to interact with robots. The application allows to set the angles of the servos, the speeds in a scale from 0 to 10 (0 being the fastest), and the time the action will take.

To reduce the number of dimensions in the project, only the angles of the servos were variable. The speeds of the servos and the timing of the actions were fixed, and in fact these factors did not affect the position of the arm since it only depended on the final values of the angles of the servos. The angles of the two servos that rotate and open and close the claw were fived too.

As mentioned in the problem statement, two databases were created for this project, one with a series of distances from the base of the robot and the corresponding angles of the 3 vertical servos (see figure 2), and another with the real angle on the plane of another series of points and the corresponding angle of the rotation servo (see figure 2).

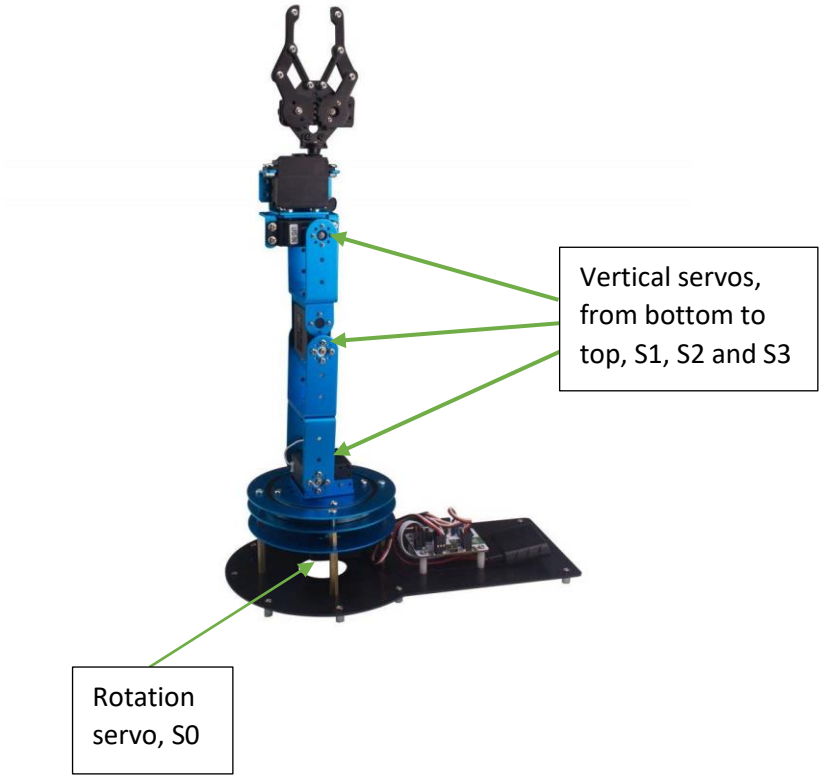

Figure 2: Rotation servo S0, and vertical servos S1, S2 and S3

The data was generated through actual trials with the robot hand, setting the angles at different values at the beginning until the first desired position was achieved, which was the first point the claw could reach on the plane without

hitting the base of the arm. This position corresponded with coordinates X = 20 cm and Y = 10 cm. From this position on, the angles were twisted manually in order to touch the plane at several points between this first one and the maximum the arm could reach on a straight line from the base, which was X = 20 cm and Y = 30 cm (X was kept fixed to obtain just the angles of the vertical servos, see figure 2). A series of 11 trials were made, reaching points at intervals of 2 cm in distance each.

A second series of trials were made, this time fixing a distance and changing the angle on the rotation servo S0 (see figure 2). In this case the problem consisted of changing the angle of S0 and measuring the point where the tip of the claw landed on the plane. Once the coordinates of the points were measured, the distance to the base of the arm was calculated using the Pythagorean theorem, $a^2 + b^2 = c^2$, where c represents the length of the hypothenuse (distance) and a and b the length of the triangle's other two sides, (coordinates X and Y of the point). Then the angle of the point relative to the base of the arm was calculated as the arccosine of the value of the coordinate X (adjacent side) divided by the distance of the point to the base (hypothenuse).

A series of 18 measurements of the angles of the rotating servo and the corresponding angles of the points were taken increasing the angle of S0 by approximately 5 degrees at a time from 0 to 90 degrees. Then the series of points from 90 to 180 was calculated (without simulations with the arm) using the same intervals between the angles of the series from 0 to 90 and applying the same correction between the angle of the servo and the angle of the point to the equivalent point below 90 degrees. For example, when the angle of the servo was 20 degrees, the angle of the point on the plane was 40 degrees. The equivalent angle of the servo from 90 to 180 was 160 (180 - 20), and the corresponding angle of the point was calculated using the same correction of 20 points between 20 and 40 degrees but subtracting it from 160. Therefore, the angle of the point corresponding to a servo angle of 160 degrees was 140 degrees.

Since one of the main objectives of the project was to create a data efficient model for robot learning, the focus was to minimize the quantity of trials required to estimate the functions relating the coordinates of the points in the plane and the angles of the servos. Applying the solution described before, the dataset of the model consisted of only 29 trials, 11 of them to assess the relationship between the distance of the points and the angles of the 3 vertical servos, and 18 with the relationship between the angle of the rotating servo and the angle of the point on the plane. There were 17 other points added to the database with the relationship between the rotation servo and the angle on the plane, but these ones were calculated, not simulated with the robot.

Exploratory Visualization

The results of the trials changing the angles of the vertical servos to reach different points in the plane are shown in figure 3. The trials show one possible solution, because there are other possible solutions for every point with different combination of angles. It is clear from the graph that the functions relating the angles of each servo to the distances are not linear. The task consisted in finding an algorithm that could capture these functions, so it could predict any possible point in a continuous space.

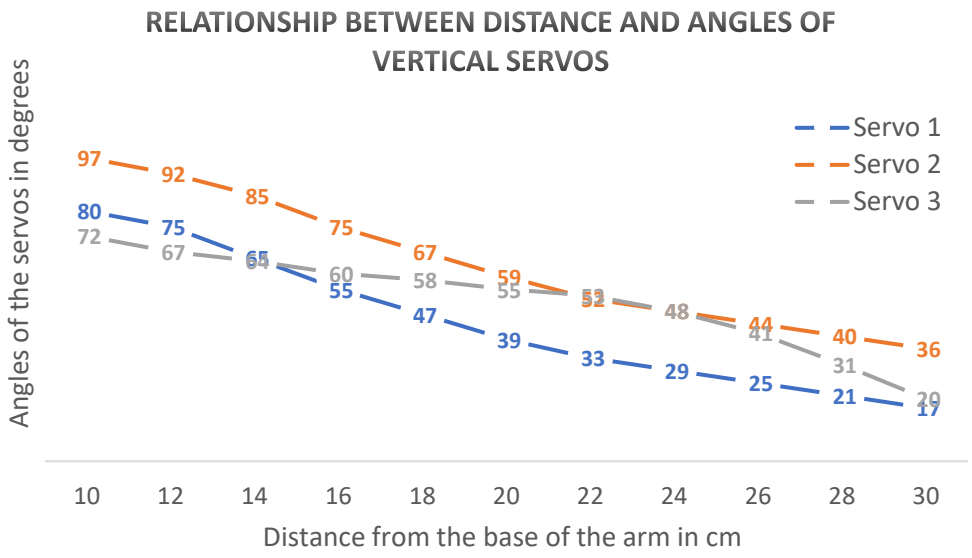

Figure 3: Relationship between distance from the base of the arm an the angles of the vertical servos

The results of the trials modifying the angle of the rotation servo is shown in figure 4. The relationship between the angle of the rotation servo and the angle of the point on the plane is not a linear relationship either. It is important to note that the position of the tip of the claw on the plane is very sensitive to the angle of the rotation servo, and this sensitivity increases proportionally to the distance from the base of the arm: the farther the point from the base the more sensitive was the position on the plane to the angle of the rotation servo.

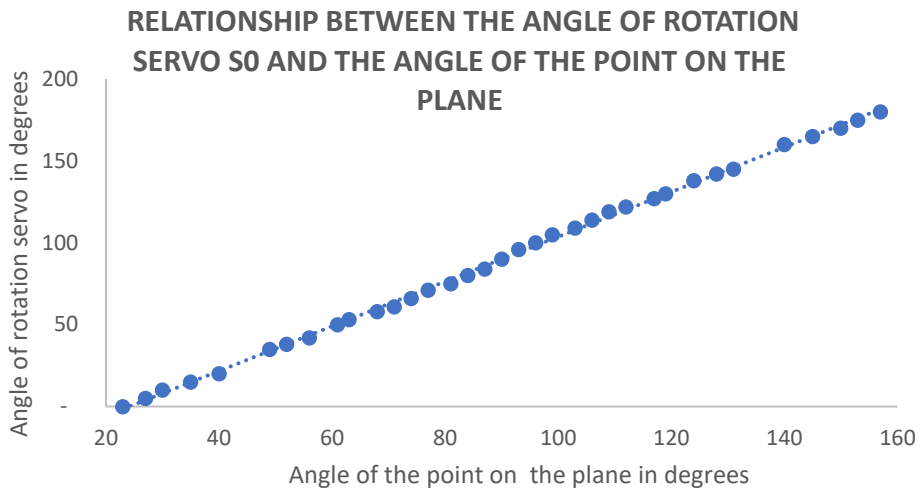

Figure 4: Relationship between the angle of the rotation servo and the angle of the point on the plane

Algorithms and Techniques

The algorithm chosen to find the functions relating the coordinates of the points on a plane and the angles of the servos was Bayesian Optimization (BO), which is a state-of-the-art model-based approach to optimization under uncertainty. BO is a global optimization method based on response surfaces [5]. Response surface-based optimization methods iteratively create a dataset D = {θ, f (θ)} of parameters θ and the corresponding functions evaluations f(θ) [5]. In Bayesian Optimization a probabilistic model is used to create the response surface that models f.

The use of a probabilistic model allows to model noisy observations and to explicitly take the uncertainty about the model itself into account, which makes the probabilistic model more robust to the effect of model errors [5]. The most common probabilistic model used in BO is a Gaussian Process (GP), and it is the one that was employed in this project.

The library Scikit-learn includes a Gaussian Process Regression (GPR) algorithm that implements a GP for regression purposes. Since each servo had a different functional relationship with the distance of the point on the plane, and with its angle, 4 GPR models were estimated, one for each of the 3 vertical servos and one for the rotation servo. Once the functions were estimated, a simulator was created in which the inputs were the coordinates of the point on the plane and the X coordinate of the center of the arm, and the result was the angle of each of the 4 servos to reach the point with the tip of the claw.

Benchmark

Two benchmark models were used to compare the results of the GPR algorithms applied to this problem. The first model used was a Lineal Regression with a polynomial transformation of degree 3, and the second was a Decision Tree Regressor. The two benchmark models were estimated using the same datasets, and the results were used in a simulator to calculate the angles of the servos for the same random points used with the GPR model.

III. Methodology

Data Preprocessing

This project used only two small datasets, one with 11 measures of the angles of the 3 vertical servos and the distance on the plane the tip of the claw reached (shape 11, 4), and another dataset with 35 angles of the rotating servo and the corresponding angle of the point on the plane (shape 35, 2). All the models to estimate the functions linking the angles with the points on the plane were univariate GPR, with the distance as dependent variable in the case of the vertical servos, and the angle on the plane for the rotation servo. The independent variable was always the angle of the servo.

Implementation

The implementation process included two main steps, as mentioned in the Problem Statement section: the fist step was to estimate the functions linking the distance on the plane to the angle of the vertical servos and the angle of the point on the plane to the angle of the rotation servo, and the second step was to create a simulator using the predict method of the GPR with coordinates of points in a plane as inputs.

All the models and the simulator were estimated in jupyter notebooks, and the algorithms used were imported from the scikit-learn library. Figure 5 shows the implementation of the 4 GPR models, where X is the distance on the plane, X0 is the angle on the plane, and y0 to y3 are the angles of the 4 servos. A noise factor was introduced in dy0 to dy3, and a stationary Radial basis function (RBF) kernel. The kernel's parameters were estimated using the maximum likelihood principle. The n_restarts_optimizer is the number of restarts performed to find the parameters of the kernel that maximize the log-marginal likelihood.

```
# Define a noise parameter
dy1 = 0.5 + 1.0 * np.random.random(y1.shape)
dy2 = 0.5 + 1.0 * np.random.random(y2.shape)
dy3 = 0.5 + 1.0 * np.random.random(y3.shape)
dy0 = 0.5 + 1.0 * np.random.random(y0.shape)

# Define a kernel
kernel = C(1.0, (1e-3, 1e3)) * RBF(10, (1e-2, 1e2))

# Instanciate the models
gp1 = GaussianProcessRegressor(kernel=kernel, alpha=(dy1 / y1),
                               n_restarts_optimizer=10)
gp2 = GaussianProcessRegressor(kernel=kernel, alpha=(dy2 / y2),
                               n_restarts_optimizer=10)
gp3 = GaussianProcessRegressor(kernel=kernel, alpha=(dy3 / y3),
                               n_restarts_optimizer=10)
gp0 = GaussianProcessRegressor(kernel=kernel, alpha=(dy0 / y0),
                               n_restarts_optimizer=10)

# Fit models to data
gp1.fit(X, y1)
gp2.fit(X, y2)
gp3.fit(X, y3)
gp0.fit(X0, y0)
```

Figure 5: Implementation of the GPR models in jupyter notebooks

The other main step in the implementation of the models was the simulator to predict the angles of the servos given the coordinates of a point in a plane. As figure 6 shows, the inputs of the simulator were the X and Y coordinates of a point in the plane, and the X coordinate of the center of the arm on the plane. Then the distance of the point to the base of the arm is calculated using the Pythagorean formula $a^2 + b^2 = c^2$ where a and b were X and Y coordinates and c was the distance of the point. The angle of the point is calculated as the arccosine of the value of the coordinate X (adjacent side) divided by the distance of the point to the base (hypothenuse).

Figure 6 shows also an example of the simulator in action. In the example the coordinates of the point were X = 15 cm and Y = 20 cm, while the center of the arm was at coordinates X = 20 cm and Y = 0 cm. Given these inputs, the simulator calculates the angle on the plane, which in the example was 104 degrees, and the distance of the point to the base of the arm, which was 20.6 cm. Then using the distance and the angle as inputs, the angles of the servos are predicted sing the GPR for each servo.

```python
# Enter coordinates of the point and center of the arm on the plane
coordx = 11
coordy = 14
center = 20

para_dist = ((coordx-center)*(coordx-center)) + coordy*coordy
distance = math.sqrt(para_dist)
cos_angle = (coordx-center)/distance
angle = math.degrees(math.acos(cos_angle))

y_pred_punto1, sigma = gp1.predict(distance, return_std=True)
y_pred_punto2, sigma = gp2.predict(distance, return_std=True)
y_pred_punto3, sigma = gp3.predict(distance, return_std=True)
y_pred_angulo, sigma = gp0.predict(angle, return_std=True)

print('Angle on the plane = ',angle)
print('Distance = ',distance)
print('Predicted angles of servos S0 to S3 :')
print('S0 = ',y_pred_angulo.round(0))
print('S1 = ',y_pred_punto1.round(0))
print('S2 = ',y_pred_punto2.round(0))
print('S3 = ',y_pred_punto3.round(0))
```

```
Angle on the plane =  122.7352262721076
Distance =  16.64331697709324
Predicted angles of servos S0 to S3 :
S0 =  [ 136.]
S1 =  [ 52.]
S2 =  [ 72.]
S3 =  [ 59.]
```

Figure 6: Implementation of the simulator in jupyter notebooks and example of predictions given a point and the center of the arm on the plane

Refinement

The first GPR models were built using the default parameters of scikit-learn for this algorithm and comparing the predicted angles with some of the trials used to generate the database of the project. Since the robot arm was noisy, and the noise factors were not introduced in the model, the first estimations were on average 4% off the real angles of the servos. After the introduction of the noise factors, the RBF kernel and the restarts of the optimizer (Figure 5), the angles of the servos used to estimate the model were predicted without errors.

IV. Results

Model Evaluation and Validation

The final model was validated with 10 trials introducing random points in the simulator and using the angles it predicted to grasp one cubic cm cubes located on top of the

points. There were 3 sets of trials with the same 10 points to control for the noise of the arm. Figure 7 shows the LewanSoul LeArm grasping a cube during a trial. Of the 3 sets, 100% of the trials were successful in grasping the cubes in 2 of the sets, and 90% were successful in one of the sets, meaning only one cube was not reached in the 30 trials done with the GPR models.

Justification

The angles of the servos were predicted also using the two benchmark models: Lineal Regressions with a polynomial transformation of degree 3, and Decision Tree Regressors. The same procedure to test the results, and the same points used in the test with the GPR model, were tested with the benchmark models. Of the 3 sets of trials with the Lineal Regressions, in two of the sets the arm grasped 70% of the cubes, and in one of the sets it grasped 80% of the cubes. The results of the sets of trials with the third model - Decision Tree Regressors- were the worst of the 3 models: the arm only grasped 40% of the cubes in each of the 3 sets of trials.

Finally, to test the reproducibility of the model, it was estimated for a different robot arm, with other measures and dynamics than the LewanSoul LeArm. It was another low-cost manipulator, branded SaintSmart, and bought for US$ 136,99 on Amazon (figure 7).

This arm was a lot noisier than the LewanSoul, and more fragile. The same procedure to estimate the models was applied with the SaintSmart arm -see steps in the Project Statement section-, and a simulator was built to predict the angles of the servos given the distance and the angle of points in a plane. The same robot controller, ezb-v4, was used to control the arm. This controller has 24 servo ports, so with both arms only half of the ports available were used.

The results of applying the GPR model to grasp cubes on a plane with the SainSmart arm were less spectacular than the ones with the LewanSoul. Of the 3 sets of 10 trials each, in 2 of the sets the arm grasped successfully 70% of the cubes, and in one of the sets it grasped 80% of the cubes. The benchmark models were also tested with the SaintSmart arm, and both the Lineal Regression with polynomial transformation and the Decision Tree Regressor performed similarly, with two of the sets of trials succeeding in grasping 40% of the cubes, and one set of trials each grasping only 30% of the cubes.

As mentioned before, this arm was a lot noisier and fragile than the LewanSoul. Some servos had to be replaced during the exercises because they burned, the joints had to be oiled due to fast tear and wear, and the bolts and nuts had to be tightened several times. But despite the noise and fragility of this arm, the

model was robust enough to have it grasp successfully at least 70% of the cubes in repeated trials.

V. Conclusion

Free-Form Visualization

Figure 7: Robot Arm LewanSoul LeArm (top right) and Robot Arm SaintSmart (bottom left) grasping cubes on the plane given its coordinates

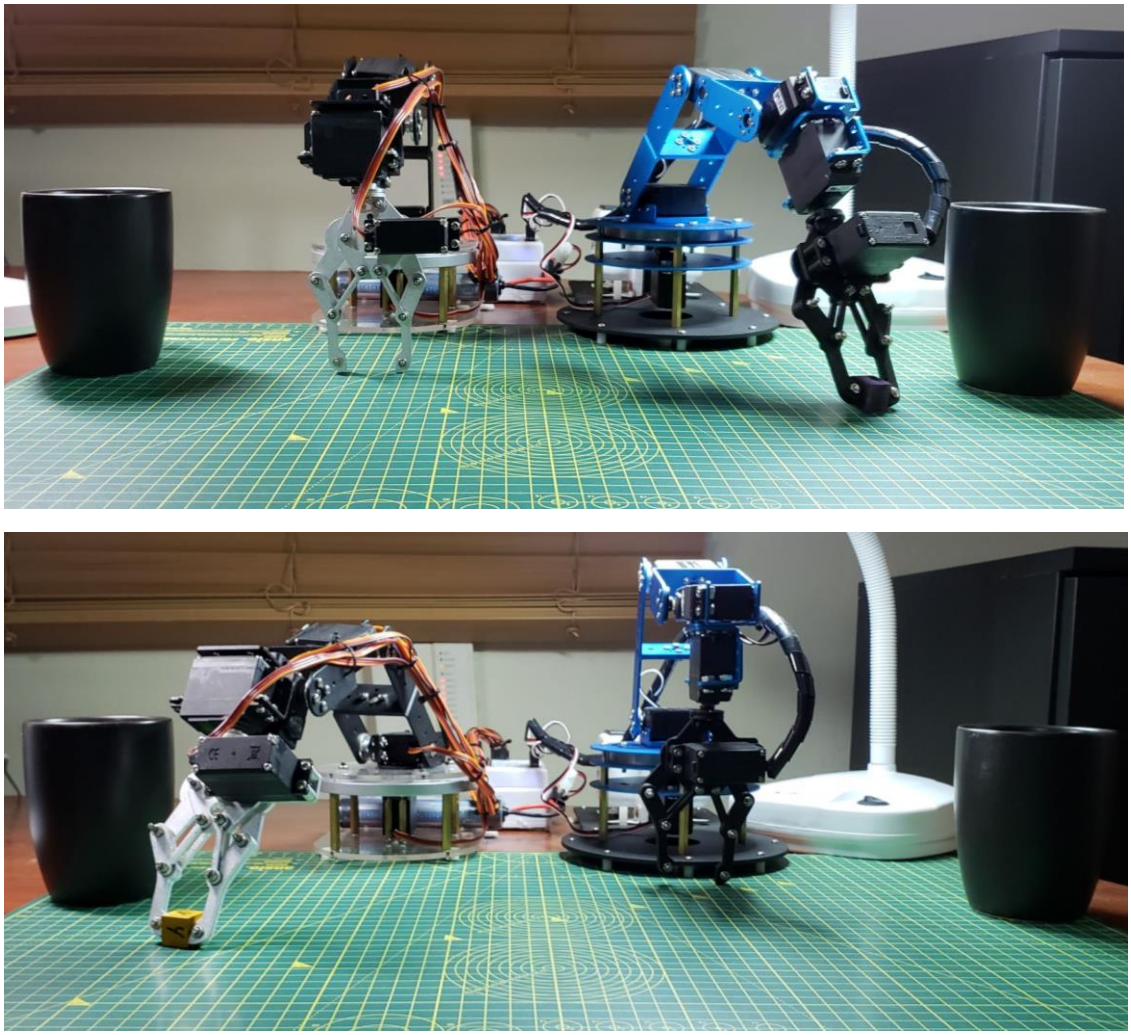

Figure 7 shows both robot arms grasping cubes during a trial, and in the following link there is a short video in YouTube with this trial: https://youtu.be/9GrqrWKiXJE. Grasping the cubes was the only success metric of the project, so it was important to show them to give a better idea of how it worked. Using bigger or smaller cubes has an impact on project metrics, and at some point in the project bigger cubes were used, and the metrics improved,

particularly for the noisier arm. However, using 1 cubic cm cubes made it easier to standardize the procedure and to facilitate the reproducibility of the project.

The project only dealt with the point in the plane the tip of the claw would reach, so the angle of the servo that rotates the claw was kept unchanged. In consequence the cubes had to be placed in line with the rotation angle of the arm or else it could not grab it. This is another factor to consider in trying to reproduce the results.

Reflection

Teaching a low-cost robot arm to grasp objects is not an easy task. However, as this project showed, one can take a robot arm out of the box with no knowledge of it at all, and still be able to teach it with a few trials to reach any point on a plane. The key aspect to consider is that low-cost robot arms are noisy, so one should build a model that takes noise into account.

In this project two low cost robot arms were taught to reach any point in a plane and grasp cubes with a minimum of trials. The data was generated in the inicial trials to reach points in the plane first changing the vertical distance, and then fixing a distance and changing the rotation angle of the arm. The dataset was uploaded in jupyter notebooks, and a Gaussian Process Regressor algorithm was imported from scikit-learn. With the database of the trials a GPR was estimated for each servo of the arm, and then a simulator was created to find the combination of angles for any point on a plane.

Even though the project aimed at using a frugal approach to robot learning, and that it was achieved with less than 30 trials for each arm, the whole process took more than a month work for several hours a day. The arms had to be assembled, controlled, and repaired. The data was read from the controller and introduced manually on Excel, where the database was created. Then is was uploaded on jupyter noteboks and the models were estimated and tuned.

Building the simulator took several hours of refreshing high school mathematics. Once it was built, the predictions for the trials had to be taken manually from jupyter notebooks to Excel one by one, and the angles re introduced in the robot controller manually too. Then the benchmark models were built and tested following the same procedure, and the whole process had to be repeated for the second arm. But however long and painful it was, the process was also a lot of fun for a robot lover.

Improvement

Some improvements that could be made to this project in the future are the following:

1. Use a program like ROS that allows a two-way communication between the controller and the robot. This would allow to apply a Reinforcement Learning approach and let the robot improve its performance by itself.
2. Try with less simulations to reach an optimum point with as little simulations as possible allowing correct predictions.
3. Generalize the model so the arm can reach any point in a 3-dimensional space instead of just a plane.
4. Include the rotation and opening and closing of the claw in the model, so it can grasp other kinds of objects with different shapes and in different positions.
5. Determine the coordinates of the points with a camera so the robot can calculate the distance and angle of the points by itself.
6. Add an object recognition model to differentiate and choose objects.
7. Add sensors to the robot to detect collisions, textures, and determine the positions of the parts of the arm at any moment.

References:

1. Deisenroth, M. P., Rasmussen, C. E., & Fox, D. (2012). Learning to control a low-cost manipulator using data-efficient reinforcement learning. In *Robotics: Science and Systems*. (Vol. 7, pp. 57-64). MIT Press Journals.
2. M. P. Deisenroth and C. E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In ICML, 2011.
3. Deisenroth, M.P.; Fox, D.; Rasmussen, C.E. (2014). Gaussian Processes for Data-Efficient Learning in Robotics and Control, IEEE Transactions on Pattern Analysis and Machine Intelligence.
4. Calandra, R., Seyfarth, A., Peters, J. et al. Ann Math Artif Intell (2016) Bayesian optimization for learning gaits under uncertainty. 76: 5. https://doi.org/10.1007/s10472-015-9463-9
5. https://www.ez-robot.com/Shop/Default.aspx?CatId=9
6. https://www.ez-robot.com/EZ-Builder/

7. Polydoros, A.S. & Nalpantidis, L. J Intell Robot Syst (2017) Survey of Model-Based Reinforcement Learning: Applications on Robotics 86: 153. https://doi.org/10.1007/s10846-017-0468-y