SEMESTER PROJECT

Improvement and Optimal Control of Babyfoot Rods

MECHANICAL ENGINEERING

ME-401

Supervision:
Dr. Christophe Salzmann

Student:
David Nigri

Laboratoire d'Automatique

EPFL STI IGM LA
ME C2 408 (Bât. ME)
Station 9
CH-1015 Lausanne

Téléphone :	+ 41 21 693 38 49
E-mail :	christophe.salzmann@epfl.ch
Site web :	https://la.epfl.ch

Lausanne, September 2019

Semester Project ME
Fall 2019

Candidate:	**David Nigri** **Mechanical Engineering Section**
Subject:	**Babyfoot: vision and control**

Le but de ce projet est l'amélioration du babyfoot existant.

La partie vision devra être revisitée afin de permettre une calibration précise prenant en compte toutes les corrections nécessaires.

La stratégie actuelle devra être améliorée pour prendre en compte le contrôle des trajectoires plutôt que le contrôle des positions comme actuellement.

Le nombre de crédits réservés au plan d'études pour ce projet est de **dix.**

Dr. Christophe Salzmann

Contents

1 Introduction **3**

2 Vision **4**
2.1 Improvements on the Vision and Vision calibration 4
2.2 Error on vision and troubleshooting 6

3 Strategy and hardware improvements **8**
3.1 General rearrangements 8
3.2 VIs Ball Stuck and Security check 9
3.3 New encoders sensors for rotation 10
3.4 Mirroring 11
3.5 Control and material failure 11

4 Optimal Control **12**
4.1 First try for MPC 13
4.1.1 MPC, description and perspectives 13
4.1.2 System Identification 14
4.1.3 Constraints 15
4.1.4 Implementation 16
4.1.5 Results 17
4.2 Necessary future improvements 19
4.2.1 Definition of the Optimization problem 19
4.2.2 Better System ID 19
4.2.3 Estimator 19
4.2.4 Different strategies 20
4.2.5 Faster MPC 20

5 Conclusion **21**

6 Acknowledgments **21**

Appendices **22**

1 Introduction

The Laboratoire d'Automatique's Baby-foot is one of the flagship projects at EPFL. It fascinates people from all ages every time it is on display, thanks to the ingenious way the lab worked over the years on this simple old sport, to bring it to the 21st century. It is also a great project for any engineer interested in robotics and automatics. So naturally, I was happy Dr. Salzmann accepted taking me for this project.

Although, at the beginning, I had big plans for improving the system and implementing ambitious solutions for the flaws in the strategy, I came to realize that it is much harder to improve on an existing project than to start from scratch. There is necessarily going to be much more troubleshooting or even just time to figure out what was done before.

Nonetheless, even though it was more of a challenge, I definitely learned a lot more when trying to figure out what was going on and what was wrong (for example, I learned a lot about connecting motors, encoders and interference, which weren't supposed to be part of the project at all), and after all, I'm glad I went through all this.

This report's aim is to represent an overview of what was done during this semester, from the improvements we actually made, to the challenges we faced and couldn't solve, and the future improvements that should be done. The first part will be dedicated to the Vision, we'll than discuss a few improvements made on the strategy, and finally we'll tackle the optimal control of the rods.

2 Vision

2.1 Improvements on the Vision and Vision calibration

After a fresh calibration of the camera's mapping (as explained in a previous report from 2018), we checked its validity in a VI called ChS_TestCalibration.vi. This VI, although quite simple, was particularly helpful because it allowed us to check very easily (just pointing it on the screen, and measuring the position on the playground) that every point on the image was mapped in the software to the right position on the Baby-Foot.

Despite the calibration (and thus, the mapping) being very accurate, when trying to play with the ball, the software wouldn't map it correctly when tracking it. In order to comprehend this issue, we created the VI "test_video_tracking2.vi" which only does tracking and is used to compare the tracked position of the ball with its real position.

We realized that the transformation on the positions that had already been added to take into account the deflection from the acrylic panel's thickness was not enough: we also had to take into account the ball's projection on the panel.
Indeed, the ball being a 3D object, it necessarily has to be projected to fit on the 2D frame the camera outputs. But because we had already mapped from the acrylic panel to the frame, it was enough to compensate for the projection on the panel.

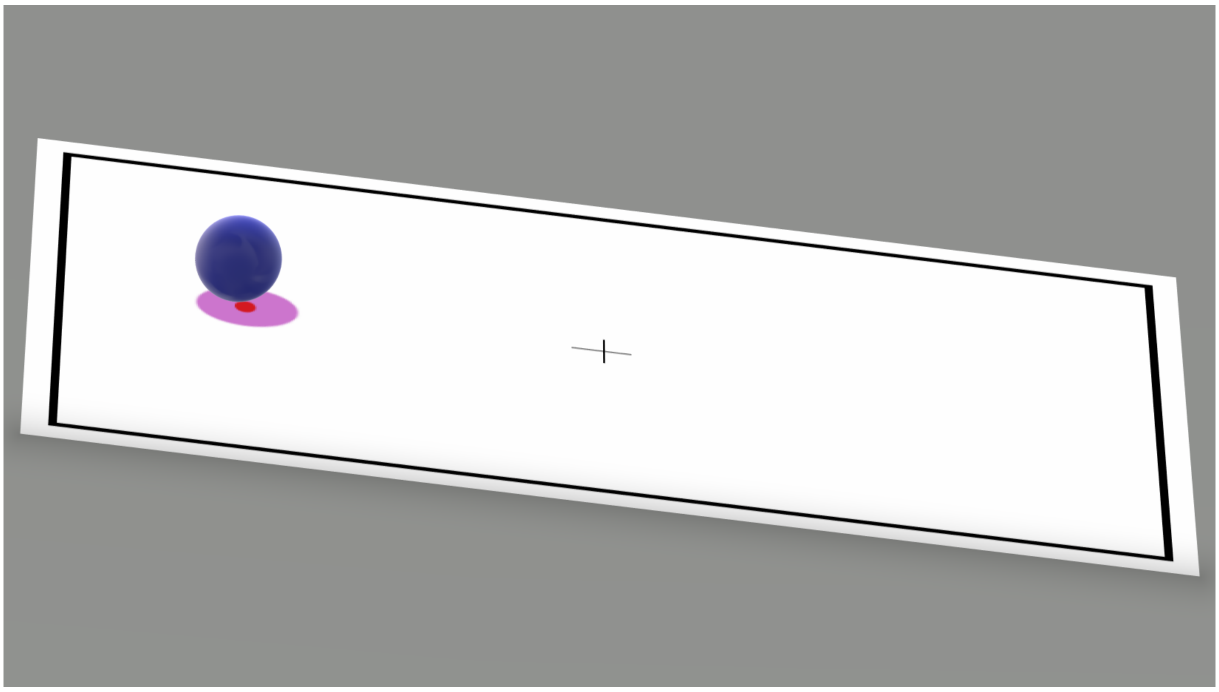
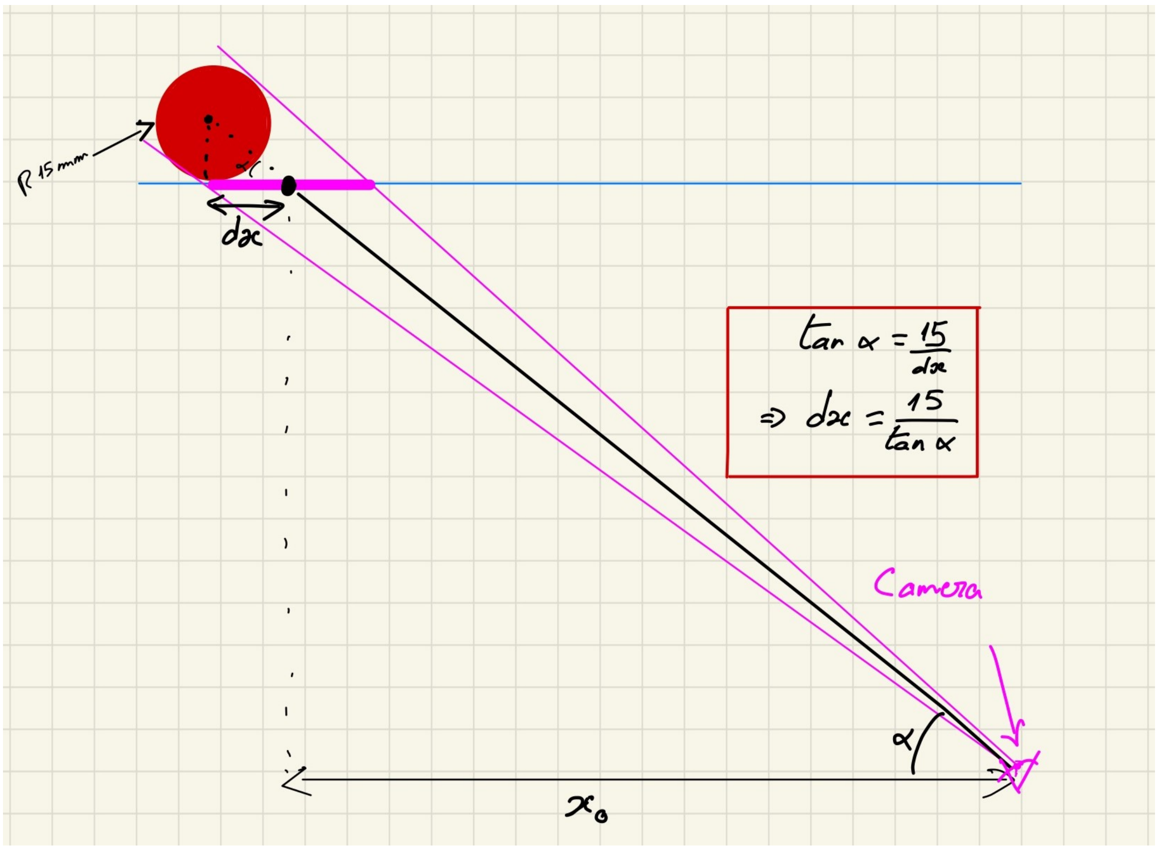


Figure 1: Projection of the ball on the acrylic panel (using Paint3D). The center of the projection (in red) doesn't match the center of the ball, hence an error in the tracking.

The issue is represented in Figure 1. We notice that the projection on the panel of what the camera reads, doesn't correspond to the exact position of the ball. Therefore, even though the mapping is quite accurate, the ball is tracked to a position close to the center of the projection, which isn't the real position of the ball.



R 15 mm
α
dx
$\tan\alpha = \frac{15}{dx}$
$\Rightarrow dx = \frac{15}{\tan\alpha}$
Camera
α
x_0

Figure 2: Geometrical simplification of the system in 2D. The transformation consists in adding a small component (which depends on alpha) to the position.

To account for this error, in a first approximation, we add another simple transformation, using some basic geometry.

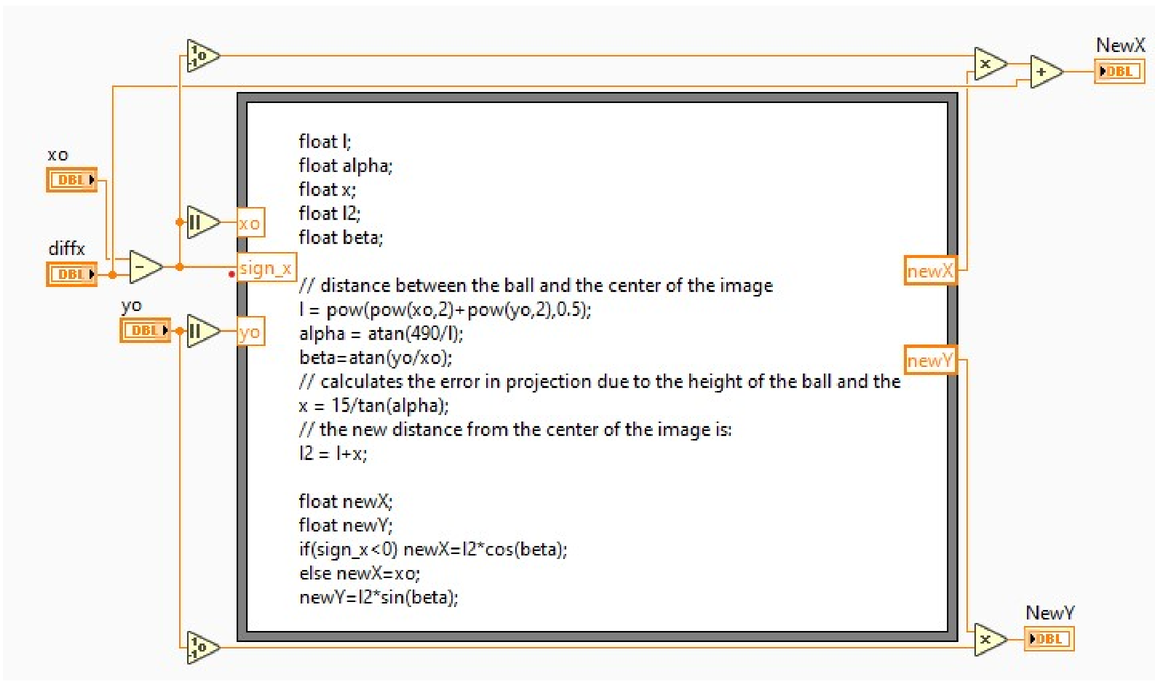
In order to simplify the case, we consider the norm of the position instead of the x and y position given by the tracking algorithm. This changes the problem from 3D into a 2D, similar to the one in Figure 2, where x_0 is the norm of the x-y position. The angle α is defined by

$$\alpha = atan\left(\frac{H}{\sqrt{x^2+y^2}}\right) = atan\left(\frac{H}{x_0}\right)$$

where H is the difference in height between the camera and the playground (which is measured to be around $490mm$). Finally, the diagram of the VI created for implementing this transformation is represented in Figure 3

Although this isn't a rigorous solution to this problem (that should imply some deeper analysis of the optics of the camera and the ball's shape), the results obtained for tracking are very good, and we finally have a precision of about 5mm (or 1 pixel) on the position of the ball.

It should also be mentioned that (as one can see in the VI) the transformation is only applied to the left half of the Baby-Foot, simply because it worked well without it, and adding the extra-component worsened the tracking.



```
float l;
float alpha;
float x;
float l2;
float beta;

// distance between the ball and the center of the image
l = pow(pow(xo,2)+pow(yo,2),0.5);
alpha = atan(490/l);
beta=atan(yo/xo);
// calculates the error in projection due to the height of the ball and the
x = 15/tan(alpha);
// the new distance from the center of the image is:
l2 = l+x;

float newX;
float newY;
if(sign_x<0) newX=l2*cos(beta);
else newX=xo;
newY=l2*sin(beta);
```

Figure 3: Geometrical simplification of the system in 1D. The transformation consists in adding a small component (which depends on alpha) to the position.

In conclusion, taking into account how imprecise the calibration process is, and how tedious it is to get a correct calibration of the mapping using ”Vision Assistant”, it would seem to me that it is not the best way to do it.

A much more efficient way to map the tracked positions to the actual positions of the ball would be by getting rid of every calibration VI used so far and start from scratch: one should start by measuring physically a few (at least a dozen) positions for the ball, and taking the values given by the tracking algorithm in Pixels when the ball is in these positions. Finally, one chooses a model of mapping (taking into account that the best ”Vision Assistant” does is a 2nd order polynomial transformation, one should probably not need to go over a 3rd order polynomial - not forgetting the cross terms like $C * x^2 * y$ or $C' * x * y$) and the coefficients θ are found with a simple least-squares method:

$$\theta = (\Theta' * \Theta)^{-1} * \Theta' * Y$$

where Θ is the matrix containing the products of the tracked x's and y's in pixels, with one line per measurement, and Y the matrix with the real positions in x and y (in mm).

2.2 Error on vision and troubleshooting

During this project, we tried to solve a recurrent problem with the camera discovered. The problem would appear systematically, at least since January 2019. When both the camera and the players were working simultaneously, after a while the camera would simply stop working. The error would appear randomly, sometimes after a couple of minutes of use, sometimes after hours. This made it very hard to understand what was going wrong because we couldn't ensure reproducibility of the error or make sure that the changes we were doing were helping or not.

So far, no solution has been found. The camera would always end up crashing after a couple of hours, but some changes seemed to work better, leaving the camera working for hours, rather than minutes.

Among them, putting away the metal structure on which the camera usually stands seemed to work well. One possible explanation was that the structure was acting as an antenna, and was conducting all the interferences from the surroundings (and the rods) to the camera. This explanation doesn't really makes sense though, because, when testing for the conductivity of the structure, it became clear that the metal was treated and couldn't conduct any electricity to the camera.
Furthermore, even when the camera was sitting on wood, the crash still happened after a while. We also tried creating some kind of Faraday cage to avoid any interference, but this didn't work either.

Another idea was that for some reason, the NI drivers couldn't handle both the camera and the motors in one VI. We thus tried using two VIs in parallel, but then again, we didn't observe much of an improvement.

One last attempt that seemed to improve the use of the camera, was leaving it out of the frame of the Baby-Foot, or even resting on the table next to it. Although a number of factors could cause this effect, and we can't be sure, the fact that there is no proper tracking in this situation might have an effect.

Once again, the source of the problem has not been found, and therefore, we couldn't find a solution yet...

3 Strategy and hardware improvements

3.1 General rearrangements

During this project, the strategy has encountered a series of modifications and improvements that aimed at making it simpler and better.
The first step was a general clarification of the Strategy VI. Indeed, after years of small improvements and additions, the VI had become quite abstruse to any new student beginning to work on the project.

Hence, The new strategy VI has been designed so that it could be understood easily by anyone starting on the project with a basic knowledge in LabVIEW: the inputs are simply the position of the ball and the positions (lateral and angular of the players). The physical constants of the Baby-Foot are stored and ordered in sub-VIs. With the outputs and the constants together, the algorithm determines what case is active and from there, it computes the outputs, the command for the motors. In figure 4 one can get an idea of how it all works with, as an example, the case where the ball was in front of the Forwarders during the previous iteration.

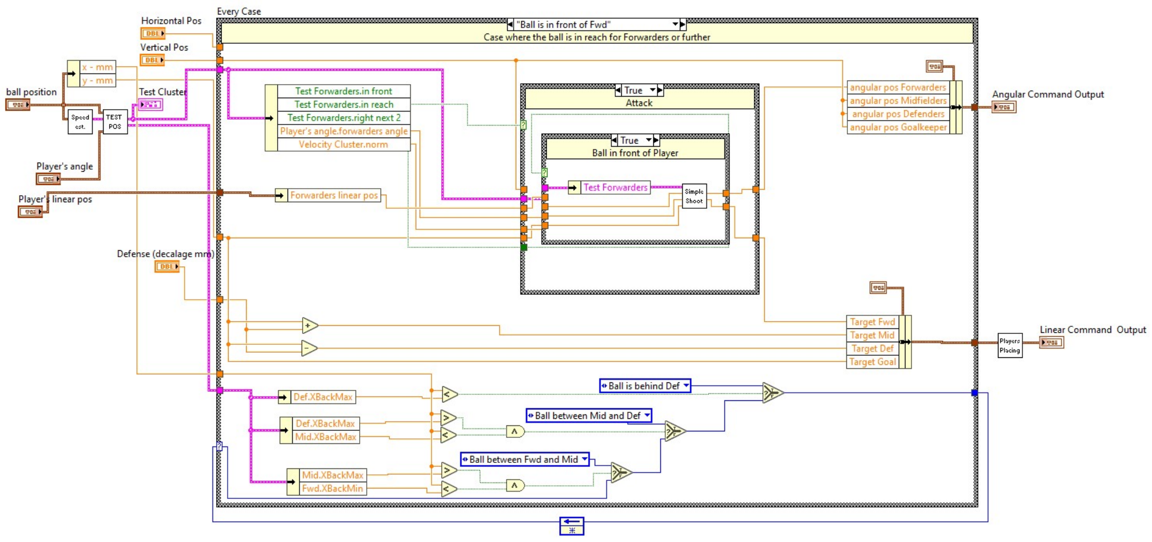


Figure 4: Diagram of the GeneralStrategy VI, with the case where the ball is in front of the Forwarders in display. On the left end, the datas of interest are gathered. In the middle, the active case is chosen in accordance with the data gathered before, and the new targets are computed for each player. On the right end, the target is mapped to a corresponding command for the motors.

As a matter of fact, the algorithm relies heavily on state machines, in order to take into consideration as many cases as possible. This is detrimental to its simplicity, as it requires internal loops and state-machines. Mainly, there is one principal state-machine to determine the general zone where the ball is. Then, there is as 2nd layer to determine whether the computer's player, in the zone of interest should "attack" or "defend", and another one to determine whether the ball is in the right position. Finally, whether the ball is in the right position or not, there is a last layer of state-machines, with, in one case, the different steps to do a simple shoot, and in the other case, move the players so that they should be in the right position in order to shoot (which we will explain further

in details, later on).
Furthermore, an important flaw of this design lays in how opaque it is, how one can't easily figure out what part of the code is active at any given time (what cases are active in the state-machines). A possible correction Dr Salzmann thought of, would be in mapping the case to different colours, that we could display on a representation of the playground in LabVIEW.

We should also mention that during this project, among the VIs written, we added a VI (called "DN_Test Position.vi") with a precise definition of the positions of every player, and the zone they cover.

In conclusion, just like in the philosophy of Science, it seems we're always looking for a trade-off between simplicity and informativeness (in this case how many cases are taken into account), and even though it might seem obvious we should prioritize the completeness over the simplicity for the best strategy, increasing the number of cases and the complexity makes it very hard for troubleshooting and figuring out what is going wrong when something is not working appropriately, which of course happens often when adding chunks of code.

3.2 VIs Ball Stuck and Security check

As mentioned before, we are going to describe, as an example, the VI "Ball Stuck". It is used when the ball is close to any of the computer's players, but the ball is behind the player. This was implemented especially not to stop the game, because without that, the ball would be immobile behind a player and the system could just stay still forever.

Once the ball is identified to be in this situation, the first step is to move the player linearly of a distance greater than the ball's diameter. Once the target position is reached (with a small enough error), we get to the next step, which consists in rotating the player, such that they would be further than the ball (we take an angle of 60°). Once the angle is close enough to 60°, we get to step 3, which consists of shifting the player so that it would be back next to the ball, but this time, ready to shoot. Finally, in the last step, the player shoots.
An additional "step 0" was also added, to act as a kind of buffer. This is particularly useful when adding a new element of strategy, to ensure that it is working as expected, and not jumping from one state to the other infinitely. It makes sure the system stays in "step 0" if the ball didn't move after the steps described previously. Therefore, it only goes back to "step 1" if the ball did move, This is evaluated by testing whether the ball's speed is below or over a certain threshold. Because our method for measuring the speed is still quite uncertain, we took the threshold to be 2, as it seemed to work well.

The diagram of this VI is represented in Figure 5.

An other example of a VI we implemented was the "Security Check". The idea is simple: just like when playing normally, the ball can in certain situations, get stuck right below a player. In this case, the error is quite big, and the integrator builds up, up to the maximum of the input. And since the ball is stuck there, the maximal input will be applied during a long length of time. This might be damaging for the motors, the rods, the structure, the players, and the acrylic (or even dangerous for the users).
Thus, this VI checks that the error on the angle on each player isn't over a certain threshold

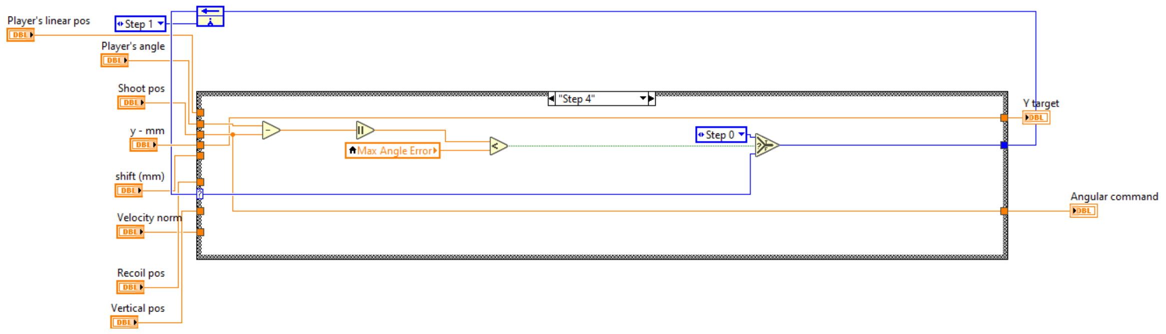


Figure 5: Diagram of the "Ball Stuck" VI, displaying the last (fourth) step.

(15°) for too long (measured in number of iterations). And if it is, it will mimic a human player, who would slightly move its rod linearly to dislocate the ball while still keeping it in it's zone of influence.

3.3 New encoders sensors for rotation

For this project, a notable improvement has been made: because of the shocks of the impacts with the ball, the coupling between the rotary motors and the rods isn't rigid, but rather done through strong friction. This inevitably induces a slip between the two items coupled at every shock.
Originally, the rotary encoders (the sensors measuring the angle of the rod) in use, were those of the motor themselves. But because of the slip induced from friction, this would result in an important shift on the angle's measure at every shock, which would deteriorate the playing experience.

We solved this issue by adding encoders directly on the rods, after the junction. The angle measured now is the one we are actually interested in: the player's angle. This implied changing the connection and the drivers in the VIs to the new encoders'.

The new encoders work well and solved the issue, but there remains an error on the forwarders. It would seem that when the rotation is too fast, the encoders miss some increments which result in an important error on the angle after couple of dozens of full turns. This could be a consequence of poor sharpness of the rising edges. Although, the fact there is also going to be an error when rotating slowly diminishes the credit one could give to this hypotheses.

Another hypothesis was that the data-sheet of the encoder was not correct and thus the number of increments per tour wasn't correct. But once again, this wouldn't explain why the error is growing with the speed of rotation.

A combination of both explanations, although a bit unlikely, could explain the errors fully.

3.4 Mirroring

Because the strategy is still a bit basic in terms of patterns and paths, We thought it would be interesting mimicking the movements of a player. In that intent, we plugged back the laser sensors, and set them up again, to measure the opponents' positions (linear and angular).

We faced an issue in that, the laser sensors measure an angle between 0° and 359° (or modulo 360), while the angle measured with the encoders, on the computer's players, is incremental and thus absolute. This is particularly problematic when the angle turns from 359° to 0° or the opposite. When this happens, the command on the motors forces the rods to a hard 360° (or rather 359°) as the error is large, even though, the players they aim to mimic, only turned 1°.

A solution found to solve this issue was doing some sort of change of coordinates. Because the discrepancy happens between 359° and 0°, the leap is quite frequent, as the players are mostly used around the vertical position. By shifting both the reflective part for the sensors and the motor's command 180°, we allow the discrepancy to happen between around 179° and 180°. This isn't a perfect solution, but it disturbs much less.

Although the lasers should be used predominantly for prediction, it would seem that trying to mimic the humans play doesn't lead to great behaviour, as the robots can go way faster, and by mimicking players, we're actually just slowing them down.

3.5 Control and material failure

The parameters of the controllers for the different motors have never been studied extensively and set up rigorously. This was made obvious when the Midfielders and the Defenders started vibrating as a resonant mode was excited. Despite lowering the gains, ensuring stability, during a day of exposition, while the Baby-Foot was under heavy load throughout the day, the rods and the motors were dislocated from the mount because of a critical failure.

This was probably the effect of the mechanical fatigue over the years on the connections. Nonetheless, it seems an in-depth analysis of the structures resistance should be ran to determine the maximum tension inputs the computer can apply on the motors without any risk of failure, and for how long.

4 Optimal Control

While trying to improve the strategy, it appeared that one important omission in the design was the control of the motors themselves.
For the angular motors, the control is done through a PID, where the parameters were not rigorously chosen, but simply set to some parameters that made the system work without vibrating.
The linear motors' regulators are computed in their drivers, and thus, the closed loop system of the motor and the regulator are considered as black boxes to which we only give the target position.
In short, we certainly don't use the motors to the best of their capacity

In addition, as we mentioned before, we hit some resonant mode with the midfielder's, defender's and goalkeeper's rods, the system suffered an important failure, and the system vibrates a lot. It seems necessary to improve on all these issues and start afresh with a rigorous controller.

Finally, because we'd like to follow particular paths, it would be a good improvement if we could control translation and rotation together.

From now on, we focus our work on the Forwarders.

4.1 First try for MPC

To control the system optimally, a simple solution would be to use a Linear Quadratic Regulator (LQR) controller. Setting the weights (matrices Q and R) appropriately, one could easily find the "optimal" input, knowing the present state. But in our case, the systems are under many constraints. Whether it is the maximal displacements of the linear motors, the limits on the input tensions to the rotational motors, or even the path we want the players to follow, classic LQR controllers cannot deal with constraints. Moreover, as the ball rarely stands still, the path we'd like the players to follow constantly changes.

Therefore, we decided to use model predictive control (MPC).

4.1.1 MPC, description and perspectives

Although LQR is optimal for a simple system, it doesn't take into account the constraints, and thus, if one wants to make sure the constraints are always met, one should restrict the initial set as much as possible, sometimes to the extent that only the origin might be possible.

MPC allows for more flexibility. Instead of relying on the very general and theoretical solution of the Ricatti equation, MPC tries to optimize the system's behaviour, form the initial condition up to infinity, taking into account the constraints:

$$\begin{aligned}
&min \sum_{i=0}^{\infty} l(x_i, u_i) \\
&s.t.\ x_{i+1} = f(x_i, u_i) \\
&(x_i, u_i) \in \mathbb{X}, \mathbb{U}
\end{aligned}$$

with l the cost function, f the function capturing the dynamic of the discretized system, and $\mathbb{X}, \mathbb{U}$ the sets for which the constraints are satisfied, respectively for the states and the inputs.

Since, obviously, no computer could possibly optimize over an infinite number, we limit the "horizon" to a finite reasonable value. But to account for the tail of the simulation, we enforce a terminal set ($\mathcal{X}_f$) for the states, a set such that if using a certain controller (for example an LQR - good for optimality), the closed-loop system remains always feasible and converges to 0. We get:

$$\begin{aligned}
&min \sum_{i=0}^{N-1} l(x_i, u_i) + V_f(X_N) \\
&s.t.\ x_{i+1} = f(x_i, u_i) \\
&(x_i, u_i) \in \mathbb{X}, \mathbb{U} \\
&x_N \in \mathcal{X}_f
\end{aligned}$$

where, to account for the tail of the simulation in the cost as well, we added $V_f(X_N)$.

The MPC method was designed to serve as an in-line controller. At each time step, we would estimate the value of the states, modify the constraints and target states, if necessary, and solve the N-horizon optimization problem. And even though, we compute the optimal inputs for the next N steps, we only keep and apply the first one (u_0), before computing starting all over again for the next step.
In this exercise though, as a first approach, we will only compute one step and apply sequentially every inputs calculated.

4.1.2 System Identification

Because it's only a first try, we wanted to keep the problem simple and use linear MPC. The optimization problem becomes

$$\begin{aligned}
&min \sum_{i=0}^{N-1} (x_i - x_s)' Q (x_i - x_s) + (u_i - u_s)' R (u_i - u_s) + V_f(X_N) \\
&s.t.\ x_{i+1} = A\, x_i + B\, u_i \\
&(x_i, u_i) \in \mathbb{X}, \mathbb{U} \\
&x_N \in \mathcal{X}_f
\end{aligned}$$

For simplicity (and because it is very time consuming), we used the transfer function computed by Ciryl Picard for the rotational motors, and the transfer function computed by Yann Morize for the "black-box" representing the linear motors and the controller. The two functions are given below:

$$G_{linear} = \frac{0.00308z^{-1} + 0.03925z^{-2} + 0.05772z^{-3}}{1 - 2.258z^{-1} + 1.829z^{-2} - 0.5062z^{-3}}$$

Equation 1: Discrete−time closed−loop transfer function of the ARX model for linear motors

$$G_{rot} = \frac{-0.002037z^{-1} + 0.01385z^{-2}}{1 - 1.9921z^{-1} + 0.9921z^{-2}}$$

Equation 2: Discrete−time transfer function of the output-error model for rotational motors

Turning them into state-space model (simply, using "tf2ss" function on MATLAB), we get

$$A_{linear} = \begin{bmatrix} 2.2580 & -1.8290 & 0.5062 \\ 1.0000 & 0 & 0 \\ 0 & 1.0000 & 0 \end{bmatrix} \qquad B_{linear} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$C_{linear} = \begin{bmatrix} 0.0031 & 0.0393 & 0.0577 \end{bmatrix} \qquad D_{linear} = 0$$

for the linear motors and for the rotational motors,

$$A_{rot} = \begin{bmatrix} 1.9921 & -0.9921 \\ 1.0000 & 0 \end{bmatrix} \qquad B_{rot} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$C_{rot} = \begin{bmatrix} -0.0020 & 0.0138 \end{bmatrix} \qquad D_{rot} = 0$$

Since we'd like to control the augmented state with states from both motors, we'd like to get the augmented system. Yet, one should pay attention to the sampling time of the transfer functions. Because of system identification considerations we won't get into, the sampling time for the linear motors' discretization is 10ms while the one for the rotational motors is 1ms. In order to use them in a common scheme, we need to hold the same input for a complete iteration of the slower system. We get:

$x(k+1) = A\,x(k) + B\,u(k)$
$x(k+2) = Ax(k+1) + Bu(k) = A^2x(k) + ABu(k) + Bu(k)$
$x(k+3) = Ax(k+2) + Bu(k) = A^3x(k) + A^2Bu(k) + ABu(k) + Bu(k)$
...
$x(k+10) = A^{10}x(k) + A^9Bu(k) + ... + ABu(k) + Bu(k) = A^{10}x(k) + \left[A^9B + ... + B\right] u(k)$
Hence, we should have $A = A^{10}$ and $B = A^9B + ... + AB + B$.

Therefore, the augmented dynamic of the system is:

$$\underline{x}^+ = \begin{bmatrix} -1.3434 & 1.5498 & -0.4392 & 0 & 0 \\ -0.8676 & 0.6155 & -0.0370 & 0 & 0 \\ -0.0730 & -0.7027 & 0.4820 & 0 & 0 \\ 0 & 0 & 0 & 1.9921 & -0.9921 \\ 0 & 0 & 0 & 1.0000 & 0 \end{bmatrix} \underline{x} + \begin{bmatrix} 1.0000 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 53.7169 \\ 0 & 44.0650 \end{bmatrix} \begin{bmatrix} u_{linear} \\ u_{rot} \end{bmatrix}$$

$$Y = \begin{bmatrix} LinearPos \\ Angle \end{bmatrix} = \begin{bmatrix} 0.0031 & 0.0393 & 0.0577 & 0 & 0 \\ 0 & 0 & 0 & -0.0020 & 0.0138 \end{bmatrix} \underline{x} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_{linear} \\ u_{rot} \end{bmatrix}$$

(when linearized, the coupling between the two systems vanishes.)

4.1.3 Constraints

Since we are using the simple linear MPC, constraints are also linear. In order to use convex optimization and find global optima, we define the constraints as a convex set. The optimization problem thus becomes

$$\begin{aligned} & min \sum_{i=0}^{N-1} (x_i - x_s)'Q(x_i - x_s) + (u_i - u_s)'R(u_i - u_s) + V_f(X_N) \\ & s.t.\ x_{i+1} = A\,x_i + B\,u_i \\ & Fx_i + Gu_i \leq b \\ & x_N \in \mathcal{X}_f \end{aligned}$$

where $Fx_i + Gu_i \leq b$ defines the boundaries of the set.

As for a first approach to MPC, we wanted to try something that was not done under the current or even previous strategy. We decided to start from a position where the ball would be on the same line as the player about 10cm away. The goal is for the player to go behind the ball, without touching it, so that it would be in the right position to shoot it. To enforce this behaviour, we define a pseudo-line as in Figure 6.

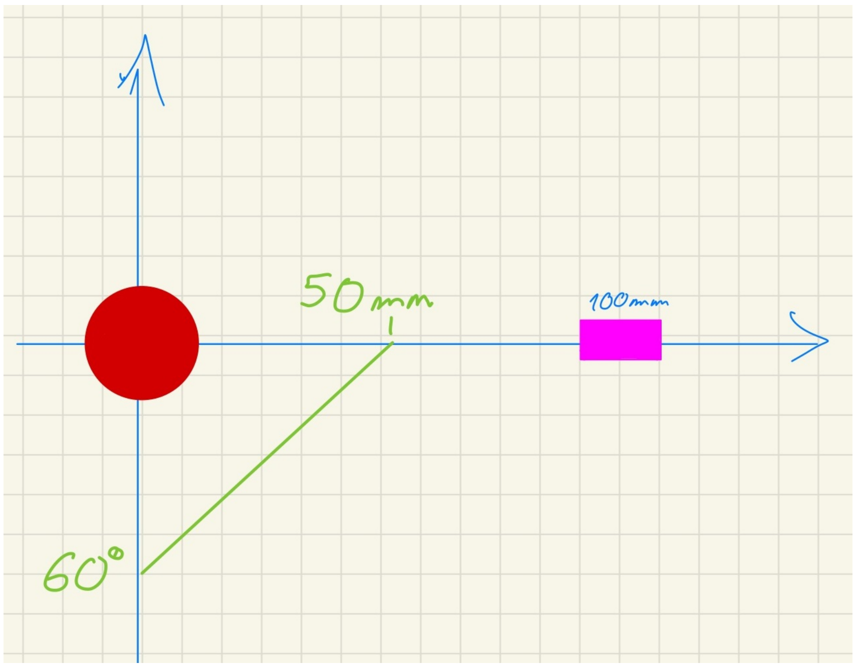


Figure 6: Outline of the simulation. The player starts vertically, 10 cm away from the ball; the ball is set on the player's line; the player should not cross the green line which defines the constraint

To avoid (too much) overshoot, we also add a constraint on the linear position

For the rotating motor, we keep the same input constraints than those considered in previous projects (+/- 10) while for the linear motors, the input is just the position and we don't need to consider them since we can start far from the boundaries.

4.1.4 Implementation

The optimization problem is solved in MATLAB, using the solver SeDuMi. For simplicity and because the system is linear with very simple state constraints, we decided to define the target point (behind the ball) as the origin of the system of coordinates. This allows us to avoid computing the target and simply require every set to be 0.

While we mentioned the MPC method allowed using terminal sets different than zero, as we are computing the input offline, we want to make sure the system reaches the target (the origin). In order to do that, while making sure the system is still feasible, we increase the horizon to allow the system to reach the equilibrium. We found that a horizon of 500 made it feasible

Finally, we set the initial condition (initial value of the 5 states) so that the output Cx_0 would be $(100, 60)$, which is the start of our path.

The code for implementing this is given in the annex.

4.1.5 Results

After running the code, we get the results in Figure 7 and 8

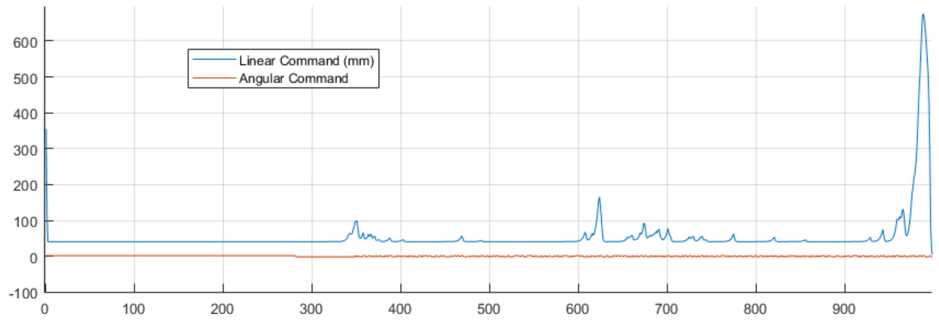


Figure 7: Command computed with the optimization problem

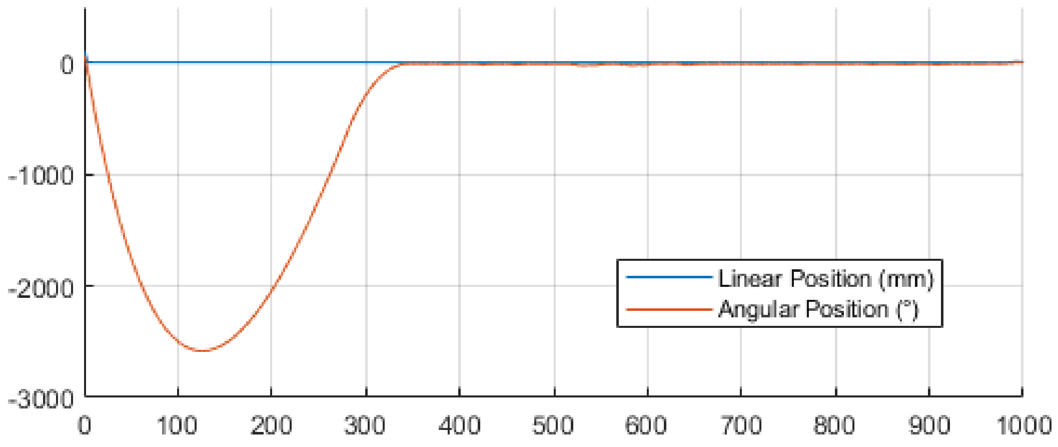


Figure 8: Resulting outputs

It appears that the results aren't optimal.

For the angular position, it seems pretty obvious that the error doesn't need to get that big! Although it could come from other issues, it is most probably a result of the B we computed to change the sampling time. Composed of the addition of 10 matrices, it probably creates an imbalance, and even a small input has too much effect on the dynamic, which takes too long to change!

For the linear system, zooming into the outputs, we get the Figure 9. Although we don't evaluate the performance or the optimality of the commands, the oscillation are an indication that something isn't going correctly.

Looking at the results, we didn't try implementing them on the physical system as we know the results would be disappointing.

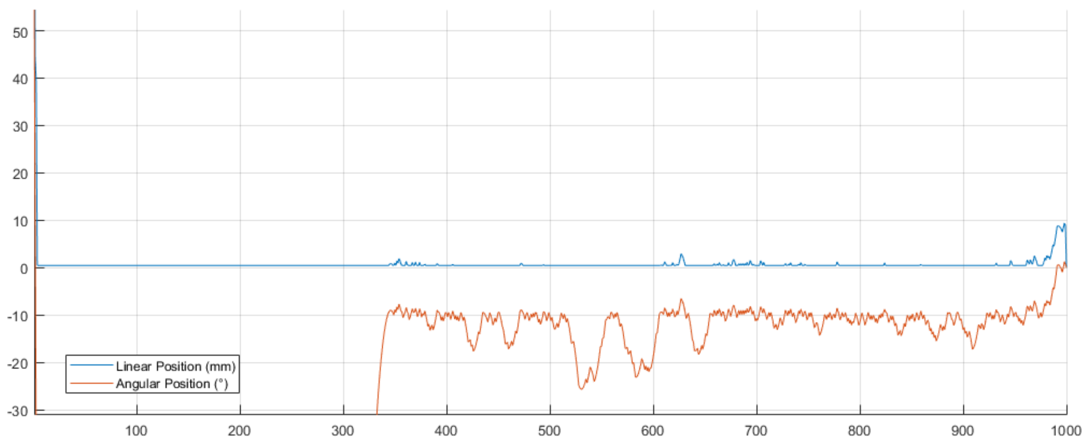


Figure 9: Resulting outputs focusing on the linear position

4.2 Necessary future improvements

Despite getting disappointing results on our first try, we still believe that MPC should be the best way to control our motors in an optimal way. Therefore, we will try here to present solutions to improve the outcome.

4.2.1 Definition of the Optimization problem

As the results in the previous part were not good, we assume there were issues with the definition of optimization problem, and the constraints.

One should give more attention to the definition of the geometric constraints, and the physical meaning of the states. Some constraints could be added, to better bound the acceptable range (not allowing more than 360° for example) or even the possible range (with the rods limited by the edges of the Baby-Foot).

Adding a terminal set along with a terminal cost would also help getting better results.

4.2.2 Better System ID

Despite the fact that we didn't get to try our commands on the real system, we know that the system identification wasn't appropriate. Primarily, the fact the two systems were not sampled with the same period, necessarily induces errors. But even independently, the system identifications weren't very reliable (with a lot of prediction error - according to Yann Morize who took care of the system identification for the linear system).

The first thing that should be done to improve the results would be to control the linear motors directly, without the help of the drivers' controller.

Next, we should take into account the non-linearities of the system: with the different frictions to be considered, the boundaries, the input constraints, the system surely isn't really linear. And although the computation is way harder, there is barely any theoretical difference between MPC and NMPC and it should be easy to implement. Hence, we would need to try the system identification with nonlinear models.

Obviously, identifying with the same sampling time seems a basic feature the two systems should have. But given that the two motors don't necessarily have similar time scales, it is not that simple. One possible solution would be to have a first regulator on each system to modify its behaviour. It would be designed so that the settling times would be the same. Then, we would control the closed-loop systems together.

4.2.3 Estimator

Because there are always uncertainties (whether it's in the model, the initial conditions, the perturbations on the state or the inputs) a good and simple improvement to implement would be an estimator. Because the MPC is designed to work in a mostly in-line fashion, it is very easy to correct the states every step.

The diagram of this system is given in Figure 10

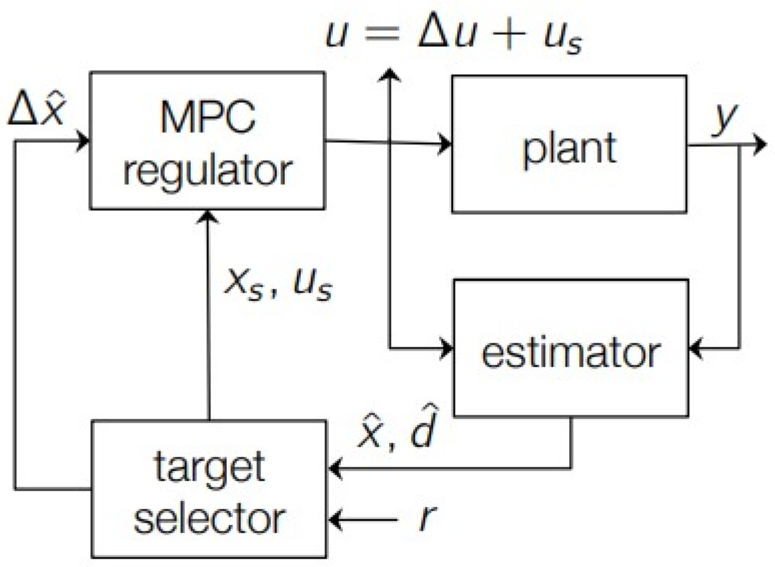


Figure 10: Diagram of a complete MPC controller, with an estimator

One could even potentially use a Kalman-Filter if we can measure approximately the uncertainties.

4.2.4 Different strategies

The power of MPC sits in how it can adapt in real time to the situation. This can be very useful when the ball is moving, and one could imagine that the constraints or the target would change accordingly.

Of course, many other paths should be added for this method to actually work, and there are different ways of programming them. For example, one could use the same method used here, creating additional constraints, or using a target that changes with time.

4.2.5 Faster MPC

Although the code for computing the optimum runs pretty fast (119ms), it is much too slow compared to the rate of the iterations of the LabVIEW code (2ms), which allows the system to react much faster.

There are methods in MPC, that were presented (but not explained) during the class of MPC given at EPFL by Prof. Jones, that allow to compute offline a map that relates current states to the optimal inputs. This would allow for a much faster MPC and should be implemented in a future project.

5 Conclusion

As mentioned earlier, it isn't simple improving on an already "older" project and some things take much more time than we would like them to.
While I wouldn't say remarkable improvements were made during this project, I believe a lot of what I did during this project will help other students in their future projects. Making sure to keep track of every change added, whether it was in this report, or in the comments of the VIs, should save them a lot of time.
The vision was also slightly improved (although we didn't find the solution to the repeated crashes), and the next calibrations should be much simpler than what had been done so far.
Finally, although the first results are disappointing, it seems to me the project should take the direction of MPC for better and more adaptive results, and we already have a good idea on what's left to do.

6 Acknowledgments

I would like to finish by thanking Dr. Salzmann for his help throughout the project and for his patience. He was always ready to try new ideas, and was very involved in every step. I also wanted to thank Pr. Karimi and Pr. Jones for their teaching which helped me understand Control better, and for being available whenever I had questions.

Appendices

```
clear all;close all; clc

ts=10e-3; %sampling time [s]
z = tf('z',ts);

%% Linear
Num_linear=[0 0.00308 0.03925 0.05772];
Den_linear=[1 -2.258 1.829 -0.5062];
[A_linear,B_linear,C_linear,D_linear]=     tf2ss(Num_linear,
    Den_linear);


%% Rotational
Num_rot=[0 -0.002037 0.01385];
Den_rot=[1 -1.9921 0.9921];
[A_rot,B_rot,C_rot,D_rot]=tf2ss(Num_rot,Den_rot);

%% Augmented Matrix
Bp=0;
for i=0:9 Bp=Bp+A_rot^i*B_rot;
end

A=[A_linear^10 zeros(3,2);zeros(2,3) A_rot];
B=[B_linear zeros(3,1); zeros(2,1) Bp];
C=[C_linear zeros(1,2);zeros(1,3) C_rot];
D=[D_linear 0;0 D_rot];


%% Setting of the optimization problem


N=1000; % Horizon

%define the constraint on x and u
%constraint of U = {u | Mu <= m}
M = [0 1;0 -1]; m = [10;10];
% X = {x | Fx <= f}
F = [-6/5*C_linear +C_rot; -C_linear 0 0; ]; f = [-10;-0.5];
    % f set different 0, to account a little bit for the ball'
    s size

[n,m]=size(B);
Q=100*eye(n); % Much more important to get a fast response
    than saving on inputs
R=eye(m);
```

```matlab

  x = sdpvar(n,N,'full');
  u = sdpvar(m,N-1,'full');
  con=[];obj=[];
  for i = 1:N-1
    con = con + (x(:,i+1) == A*x(:,i) + B*u(:,i));
    con = con + (F*x(:,i) <= f) + (M*u(:,i) <= m);
    obj = obj + x(:,i)'*Q*x(:,i) + u(:,i)'*R*u(:,i);
  end
  con = con + (x(:,N)==0);

  % Compile the matrices
  ctrl = optimizer(con, obj, sdpsettings('solver','sedumi'), x(:,1), u);

%% Simulation
x0=[0 0 1.7325e+03 0 4.3321e+03]';
[u,isfeasible]=ctrl(x0);
sol.x(:,1) = x0;
for i=1:N-1
    sol.x(:,i+1)=A*sol.x(:,i)+B*u(:,i);
end
if isfeasible == 1, error('Error in optimizer - could not solve the problem'); end

y=C*sol.x;
figure, clf, hold on, grid on
plot(y(1,:),'displayname','Linear Position (mm)')
plot(y(2,:),'displayname','Angular Position[U+FFFD]')
legend show

figure, clf, hold on, grid on
plot(u(1,:),'displayname','Linear Command (mm)')
plot(u(2,:),'displayname','Angular Command')
legend show
```