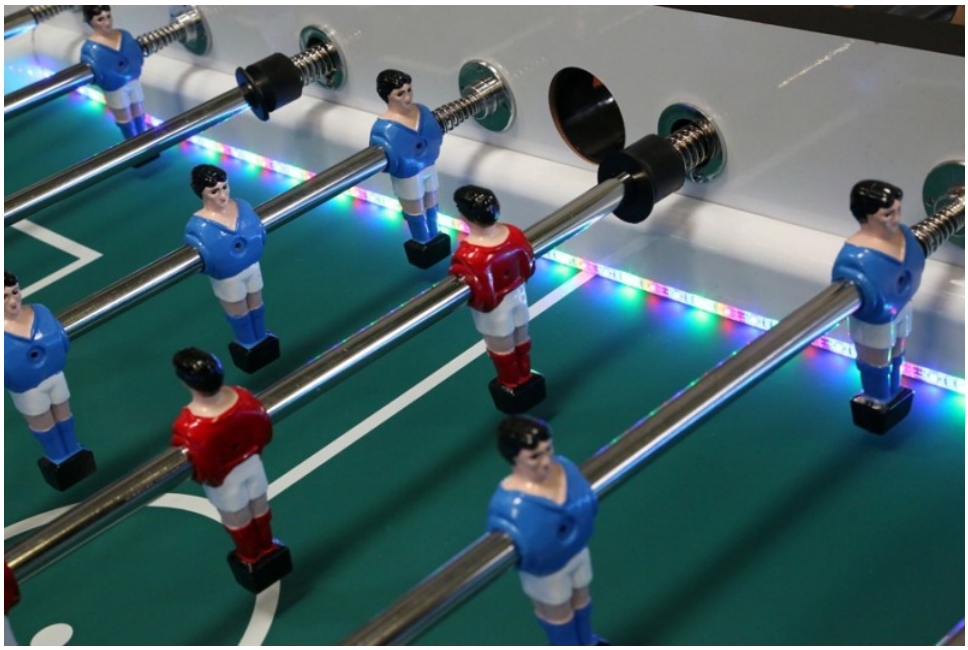
EPFL

Projet Semestre Master
Professeur: Christophe Salzmann

BABYFOOT 2021

David Kwakye



Contents

1 Introduction

With the emergence of new possibility of action of the Babyfoot and its increasing complexity, it became more and more necessary to have a way to implement everything without changing all the code every time and a place where we could get most of the information. This work wants to propose an architecture which allows to code strategies without altering the code, localise the information and allow an easy way to implement filter for measurement.

On the first month of this semester project, Mr Salzmann and the author tried successfully to resolve the problem of angular offset of the forward player. It seems like some connection was not made correctly and somehow was messing with the measurements. Future work on this Babyfoot, shall improve the implemented actions, implement some more actions and improve the control. Indeed, the author saw that linear control can achieve error to 1 cm high ! Angular error is less problematic, the max error seems to be around 2$\sim$3 degree.

2 Definition architecture project

2.1 Old way

In previous generations of the Babyfoot, there was no fixed architecture. Everything was put explicitly and it was very difficult to make any partition. This had multiple consequences for future work :

- The last version required a high understanding of the entire code to be able to do anything.
- Implementation of new strategies would have made the code way heavier. Which makes every next generation less accessible.
- Making a strategy independent is nearly impossible and therefore, altering previous strategies may require to rethink the entire code.
- The debugging becomes more and more difficult with time and require the implementation of new tools constantly
- Information are not localised and therefore, any correction becomes very difficult.

2.2 New architecture of control of the Babyfoot

We suggest an another way of doing thing. Here is how it works :

All the important constant of the Babyfoot are written as constant in the VI constante.vi and all measurements are put together in the cluster measurement. The user provide a text file with a compilation of the strategy that he want to applied to the Babyfoot. This file is interpreted once at he beginning of the program by the VI stratGen.vi and send 4 table of cluster which follow this structure :

- CI: Cluster $n \times o \times e_{no} \times 8$ with all the data of initial conditions. n is the number of implemented strategy.o is the number of condition linked by a "OR". e_{no} is the number of basic condition linked by a "AND" for the condition of the strategy currently considered. 8 is the number of parameter needed for a basic condition. Example : the third strategy has an initial condition like this [Bcondition1 && Bcondition2 || Bcondition3]. Then, to access the Bcondition3, the index needed are 2x1x0. More information in the section about Strategy
- SC: Cluster $n \times s \times o \times e_{sno} \times 8$ with all the data of success conditions. n is the number of implemented strategy. s is the number of step. o_{sn} is the number of condition linked by a "OR" for the step. e_{sno} is the number of basic condition linked by a "AND" for the condition of the step of the strategy currently considered. 8 is the number of parameter needed for a basic condition.
- AC: Cluster $n \times s \times o \times e_{sno} \times 8$ with all the data of abortion conditions. It follow the structure of SC.
- data: Cluster $n \times s_n \times 4 \times 4$ contains all the data of actions. n is the number of implemented strategy. s_n is the number of step in the strategy currently considered. More information in the section about strategy

Then, those 4 table are used by the VI Strategy.vi along with the cluster measurement, to select the strategy needed for the situation, manage the step and send a table 5x8 with the current action that need to be achieved for every row.

Finaly, the table 4x4 is interpreted by the VI Action.vi which will activate a VI according to the action that was requested and create a cluster of command for the motors.

As you can see, this method is very adaptable and powerful. Once an action is implemented, we can request whenever we want and as much as we want and make combination without altering the code. Just by changing the text file. Every important step are well define and localised. The decision making process is in strategy.vi and the action is in Action.vi and even in Action.vi, each action are implement with a vi see Action.vi which make the code way accessible since you don't need to know all the code to implement new action or just modify one. Also, since all important constant are localised in the VI constante.vi, changing values is not a problem anymore. And finally, since all measurements are localised in the cluster measurements, all you need is a filter upstream to upgrade the measurement for the entire code.

3 Inner working

3.1 Structure of the strat file

3.1.1 Syntax

The idea behind this new way of implementing requires to, more or less, create a new computer language for the Babyfoot, to indicate which kind of condition should be met and what kind of action needs to be done. It is a huge work and unfortunately, I can not pretend it is perfect yet but all the base was done during this project. It just needs some tuning.

3.1.1.1 Strat

To code a strategy using this work's syntax, instructions are supposed to be put exactly as follow:

1. *#Type of strat*
2. *Initial condition*
3.
4. *Sucess condition*/*Abort condition*
5. *Action*
6. *Sucess condition*/*Abort condition*
7. *Action2*
8.

There is 2 types or strategies. Defense and Attack. Defense strategies must be put before Attack strategies and it is very strongly advised to put as last defense strategy the Basic Defense strategy:

1. #Def
2.
3.
4. Pos5BP0Y0:/
5. GokDepB+0:+0;DefDepB+0:+0;MidDepB+0:+0;FwdDepB+0:+0;
6.

The order of the strategies is really important because it will create the hierarchy of the plan. The process of selection is explained in figure 3.1

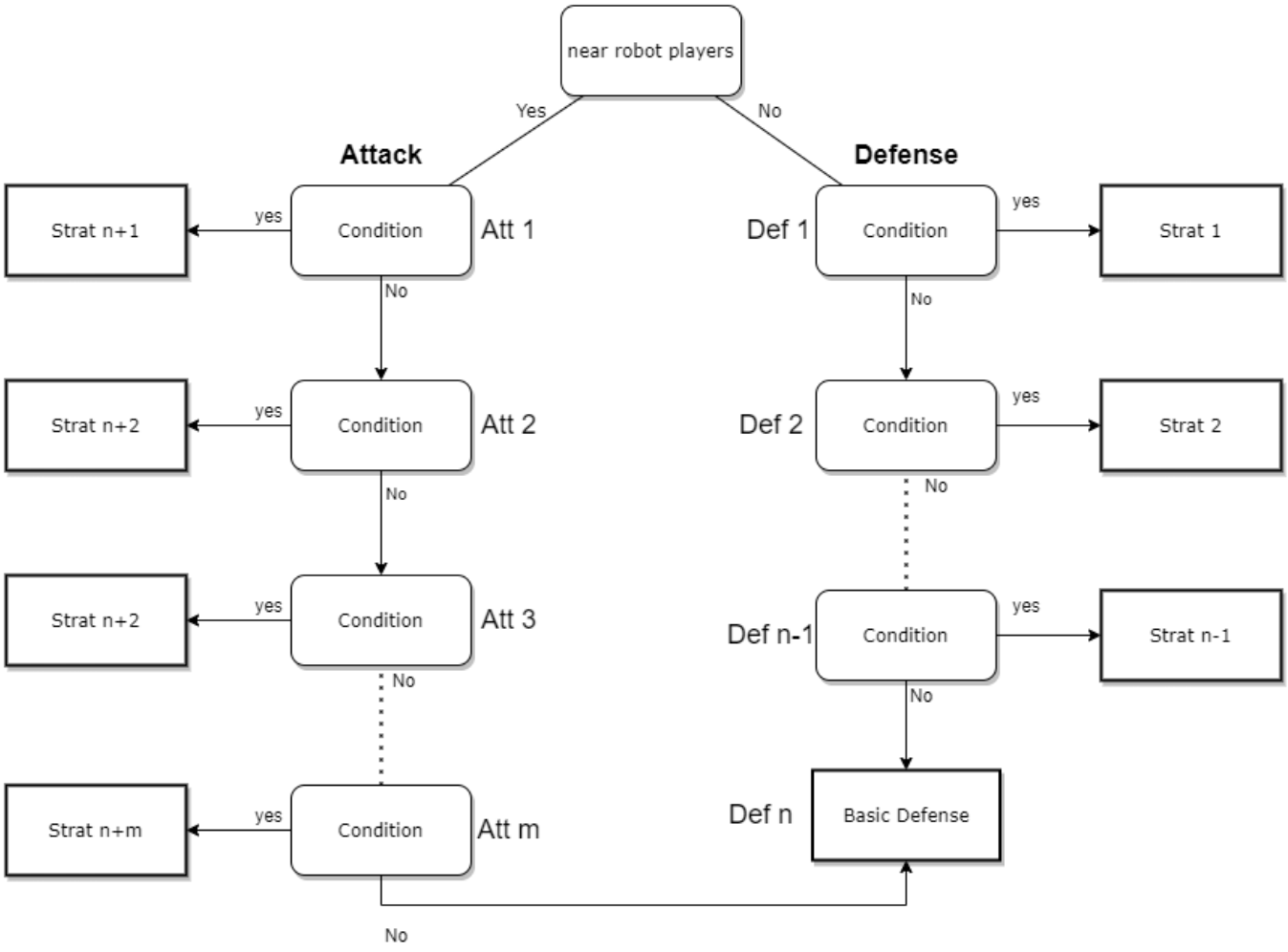


Figure 3.1: Strategy tree. First, the algorithm will look where is the ball. If it is at reach of the robot team, it will look for an attack strategy. Else, it will look for a defense strategy. It will test the current type of strategies until he find one that is true. **It will not test beyond that** which mean, it will stop looking as soon as it find a satisfied initial condition. If it does not find a satisfied initial condition, it selects the **last strategy of defense** which is supposed to be Basic Defense.

3.1.1.2 Conditions

Conditions are essential for any strategy. When we create a language, there is a trade off with complexity and freedom. Everything is not yet possible. For example, it is not yet possible to test the whereabouts of every opponent player at any time. Not because it is really difficult to implement but because the author did not have time to do it.

The conditions follow the syntax in Table 1 with the option listed in Table 2.

Pos	[0-5]	P/B	A/P	[0-5]	X/Y/T	value1 :	(value2 :)
Vel	[0-5]	X/Y/V	0	0	0	value1 :	(value2 :)

Table 1: Condition parameter

Pos/Vel	indicate if the condition is about position or velocity
[0-5]	indicate the inequality (see table 5 in annex for more precision)
P A	Take the absolute position of the robot players.
P P	Take the position of the nearest to the ball human's player with respect to the considered robot's player.
B A	Take the absolute position of the ball
B P	Take the position of the ball with respect to the considered robot's player.
X/Y/T	indicate the axis considered [mm]. T indicate an angle [deg].
value1	Mandatory value. 777 makes the conditions always true (except for abortion condition were it's always false).
Value2	Only necessary for inequalities 0 and 1, it states the upper value. 777 makes the conditions always true (except for abortion condition were it's always false).
X/Y/V	indicate the axis considered. V indicate the modulus of the velocity.[mm/s]

Table 2: Condition parameter

It is possible to couple condition together using "&" for "AND" and "|" for "OR". There is no "NOT" and every expression **must** be developed. (Condition1 | Condition2) & Condition3 is not valid for example.

3.1.1.3 Action

Action can be easily implemented. The syntax is composed of an index and 3 values which compose the options (Table 3). It is totally customable. To separate different action in the same step, a ";" needs to be put between 2 action (see next section for example).

3 letter of the Row	3 letter of the action	(Value1)	(Value2)	(Value3)

Table 3: Action parameter

At the time of this report, the implementation of action is stated in the Table 4.

Index	Action	(Value1)	(Value2)	(Value3)
0	: maintain the current position.	x	x	x
1	**Fei** : Does nothing right now.	✓	✓	✓
2	**Dep** : Displaces the row linearly (y) or rotate (ϕ) .Value1: indicate the origin of the y (Absolute A, Joueur J or Ball B). Value2 : y. value3: ϕ.	✓	✓	✓
3	**Pas** : Still in work. Pass the ball to teammate. Value1: which teammate. Value2: what kind of pass.	✓	✓	x
4	**Tir** : Shoot the ball straight forward.	x	x	x
5	**Amo** : damps the ball's velocity to capture the ball.	x	x	x

Table 4: Current action implemented. **Attention !** numerical value for Pas and Tir need to have a sign! (+/-)

3.1.1.4 Example

In this section, The Basic defense strat is explained as an example.

1. #Def
2.
3.
4. Pos5BP0Y0:/
5. GokDepB+0:+0;DefDepB+0:+0;MidDepB+0:+0;FwdDepB+0:+0;
6.

The line 1: indicate that it is a defense strategy.

The line 2: indicate the initial condition. Basic defense is a special case: it does not have an initial condition because it is suppose to be the last resort in term of defense and strategy.

Line 3: separation between the initial condition and first step.

Line 4: Conditions of the first step. The success condition is Pos5BP0Y0: which mean "if the difference between the coordinate y of the ball and the y coordinate of the nearest players to the ball is equal to 0". There is no abort condition because it is the last resort.

Line 5: Action. They are only displacement action for each row. It ask each row to go at the origin which is the y coordinate of the ball (B).

The end because there is only one step.

3.1.2 Strategy generator

Since the author was aware that it could be difficult, confusing and challenging to code long plan in a bloc note, a basic interface has been implemented : Strat_David.vi. This interface will help for the code. Although, the user must stay vigilant and **check the positioning** of each portion of code after validation. The author can not guaranty that it is bug free.

3.2 VI Strategy.vi

We can see in figure 3.2 how the decision is made in the VI Strategy. First it looks to know if the ball is within reach of the robot player. This determines if the algorithm looks for attack strategies or defense. It will then test the condition initial of the each strategy until it finds the first one which is satisfied. If it looks for an attack strategy and does not find any satisfied initial condition, then it will execute the last defense strategy. Then if the success condition is not met and the abortion condition is not met either, it will send the current action. If the condition of success is met, then it will go to the next step. If the abortion condition is met, the program comes back to chose the strategy.

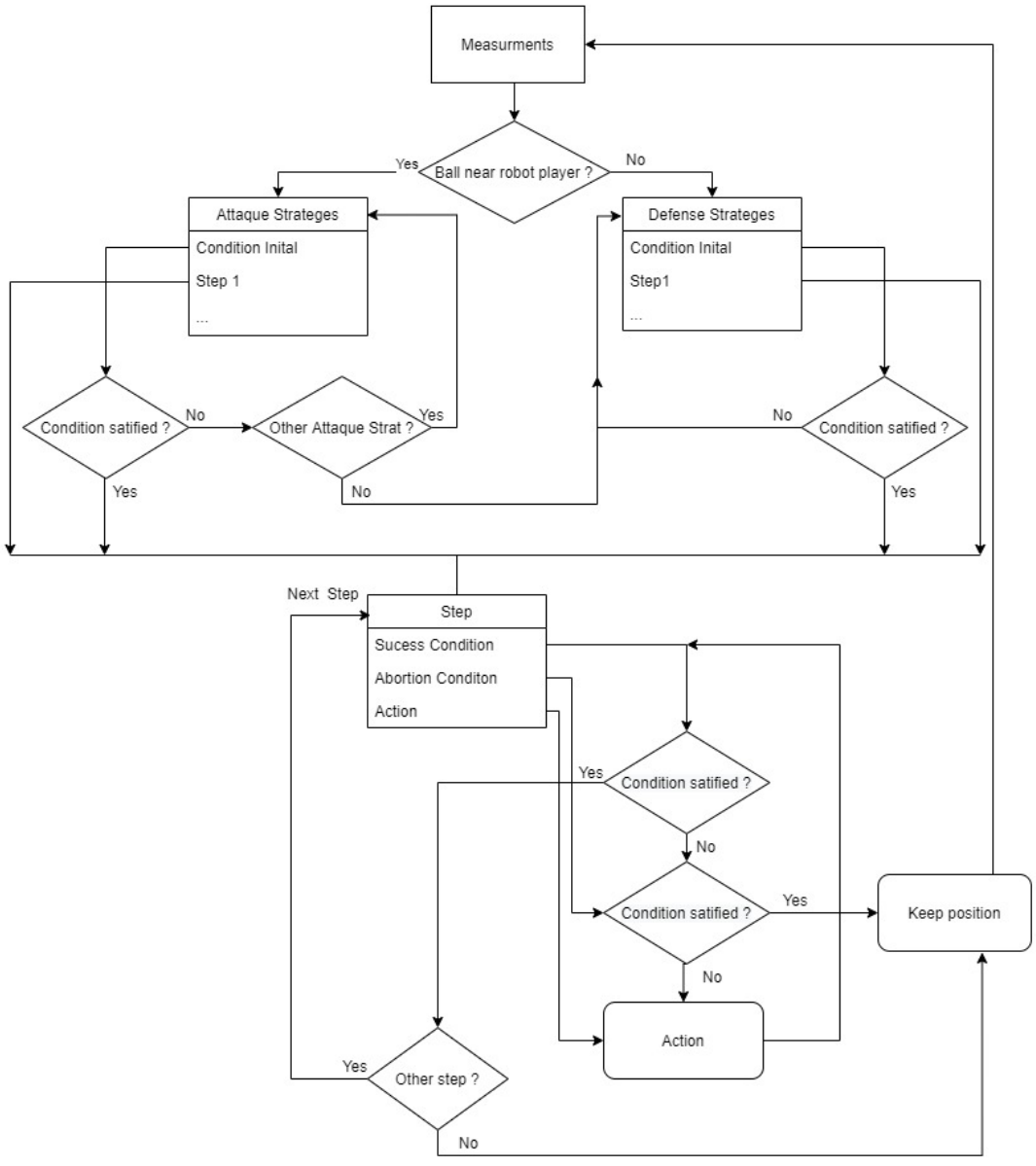


Figure 3.2: Flowchart of Action decision

3.3 VI Action.vi

In this section, a more precise description of the action handling system is provided. First, the relation between the strategy.vi and action.vi is shown in the figure 3.3.

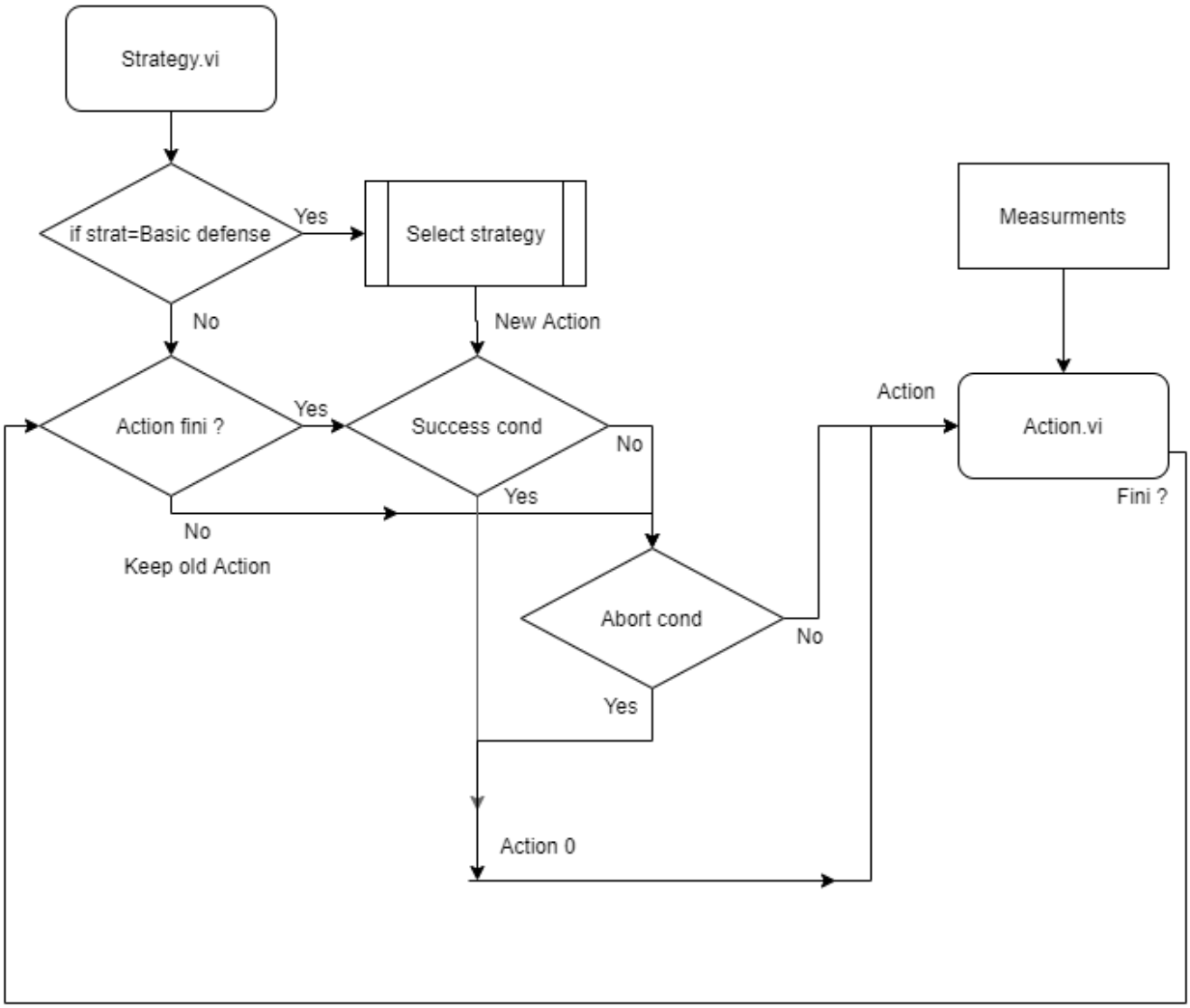


Figure 3.3: Relation between strategy.vi and action.vi

3.3.1 Principle

An Action is a table 4x4. Each row contains the data for an action for one row. The first row is for the goal keeper, the second for defense etc. The first value in the column contain the index of the action and the 3 other are the options. When the Action is send, it will go to a case structure where VI are implemented for the given action (figure 3.4).

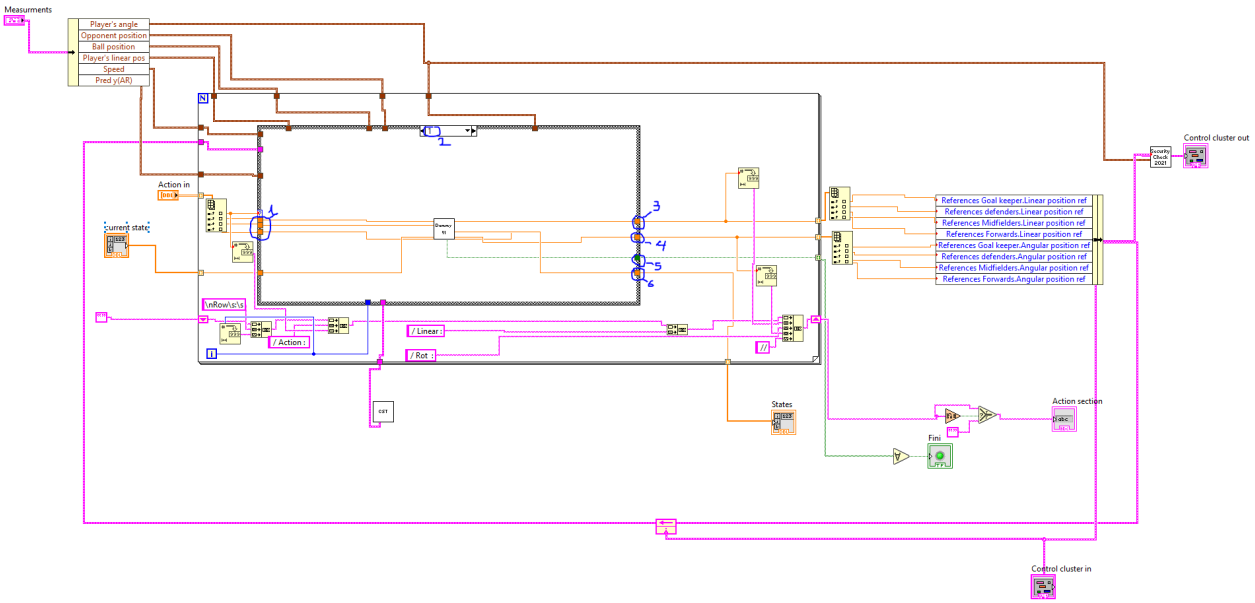


Figure 3.4: 1: Options of action. 2: Case of action. 3: Output linear position. 4: Output angular position. 5: Boolean value.It is true when the action is finished. 6: State of the action. It is advised to code the VI of the action as state machine.

3.3.2 Implement new action

In this section, the author explains how to implement a new action if the action can not be write as an option of previous implemented action or when there is no option spot available.

To keep the coding interface useful, it is necessary to keep it updated. It makes future work easier. So open the VI Strat_David.vi.

step 1 Go to to the tab New step and do a right click on the tab control *Setup* and add a new tab after. Rename the tab according to your action.

step 2 Do a right click on each enum on the left of the tab control and choose edit item. Add your action in last place.

step 3 If you have options. Create a cluster in your action's tab where you will put your controller. If you do not, go to step 5

step 4 Open WriteAction.vi and readAction.vi. Copy/past your cluester convert it to a string in WriteAction and then convert this string into an array in readAction. Check that, everything that is write in WriteAction, is readable in readAction.

step 5 Connect your cluster to WriteAction.vi in Strat_David.vi (case 3 in the case structure).

step 6 Open Action.vi and add a case to the case structure for your action.

step 7 create your action VI and it's good to go.

4 Condition testing

Condition are the base of this work proposition. But the author can not guaranty that condition are perfectly implemented. Therefore, here is proposed a way to debug condition if a problem is suspected. Be aware that the problem may come from measurements and not from the condition.

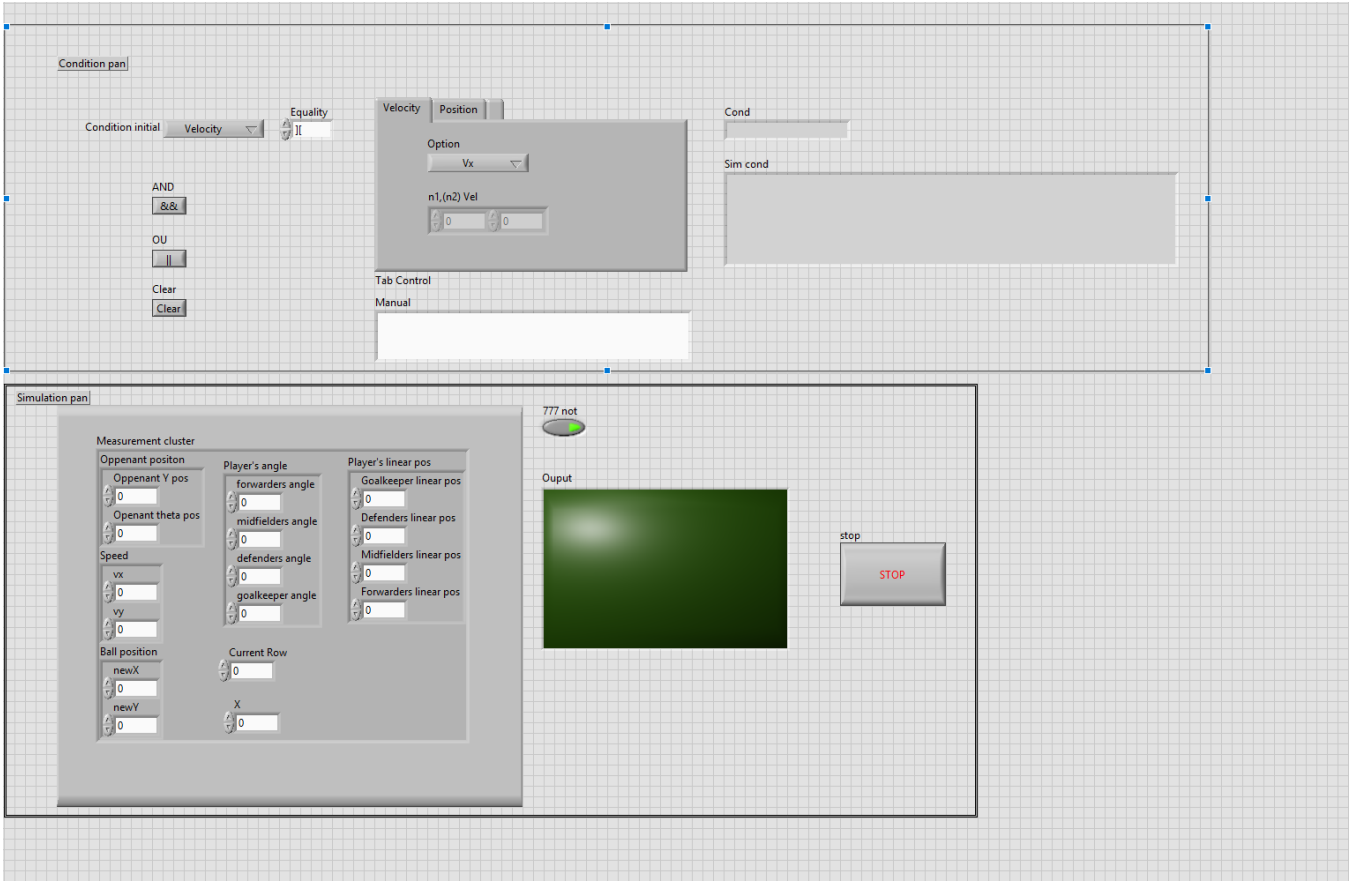


Figure 4.1: Interface Test_cond.vi

If you come across a situation were you think you do not get what you were suppose to have, open the VI Test_cond.vi in the TestVIs folder. You will get a simulator to check if there is really a problem in the condition handling system or if it's somewhere else. Write your condition and fill the cluster measurement as much as needed for your situation. The button "777 not" simulate a normal condition(IC or SC) when off and an Abort condition when on.

5 Important SubVI

5.1 Virtual vision

The centralisation of the information with the cluster measurement and the VI constante.vi made the previous virtual vision not compatible. So everything needed an update to use be used effectively. All added features of previous year have been kept and an option of recording have been implemented. The principle of utilisation is quite simple. We store in a table the measurement information and the row of each pass in the vison loop, from the moment the "recording" setting was put. And then we used them when the "lecture" setting is put.

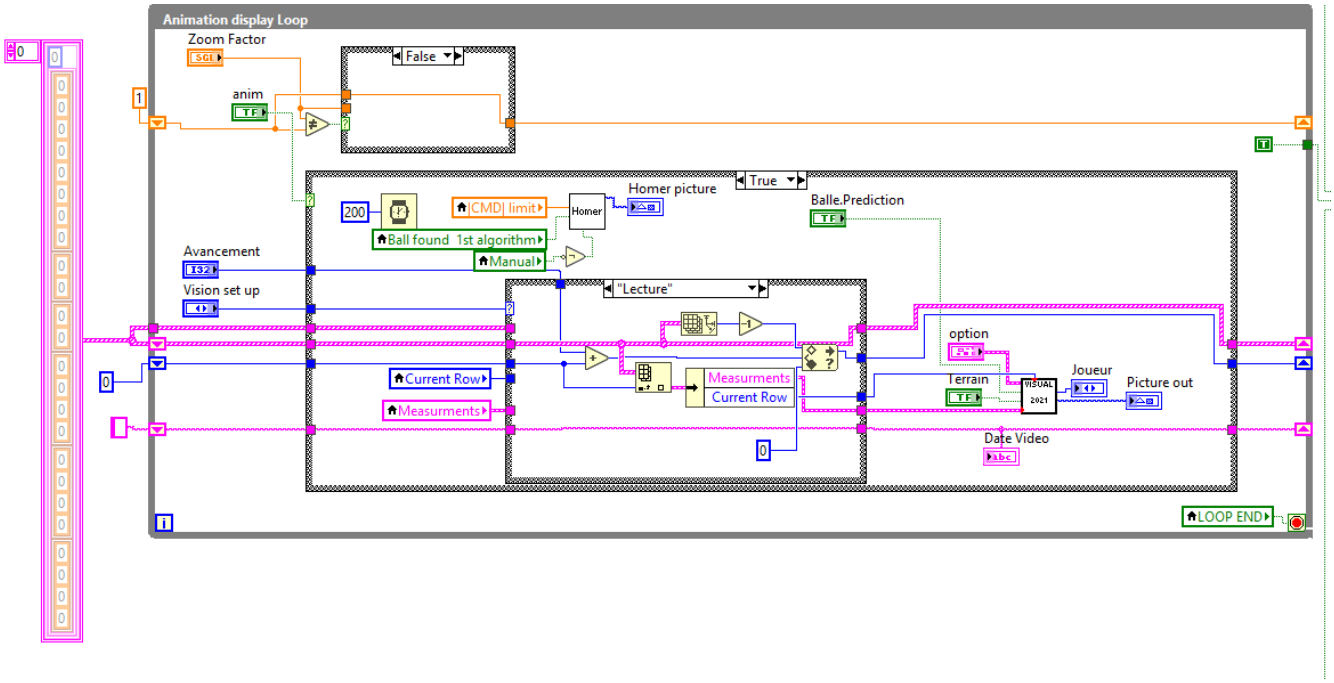


Figure 5.1: Virtual vision handling

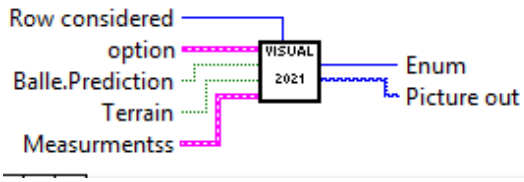


Figure 5.2: Vision 2021. Draw the situation described by the measurement cluster

Be aware that the table will rest every time the mode "streaming" is on.

5.2 Journal

To be able to see what happens in the code, a Journal have been implemented in a previous year. A VI version of it is now available. The VI reporter.vi will take a string and will put every thing in a txt file.

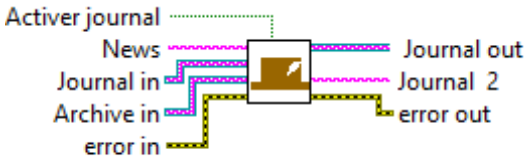


Figure 5.3: Reporter.vi

5.3 Constant

This VI is a major improvement in the functioning of the Babyfoot. This VI localises all constant of the Babyfoot and shall be used as much as possible for values which can be used at

multiple place. With that, if a value need to be changed, we change it once and it is updated for the all code.

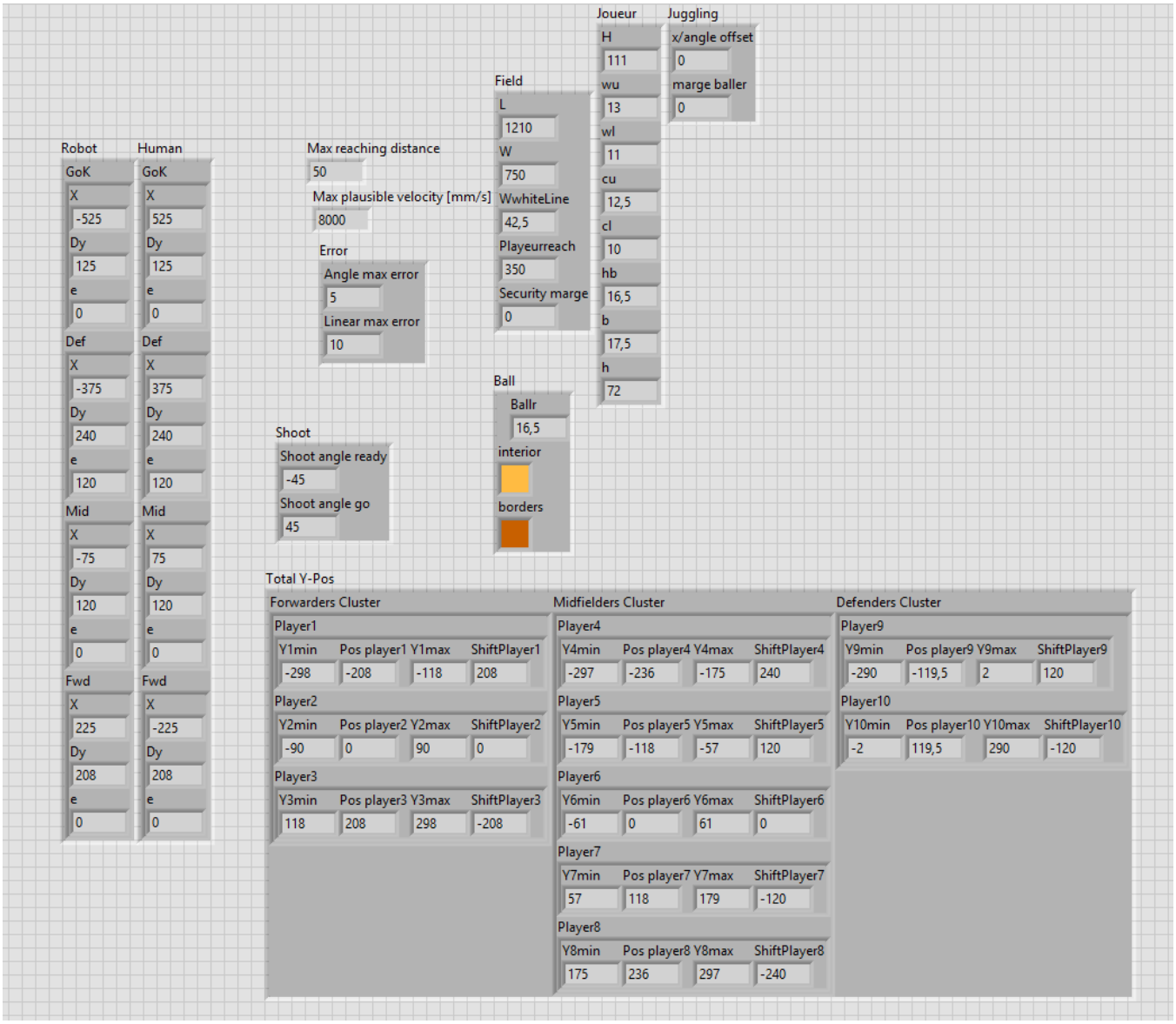


Figure 5.4: constante.vi

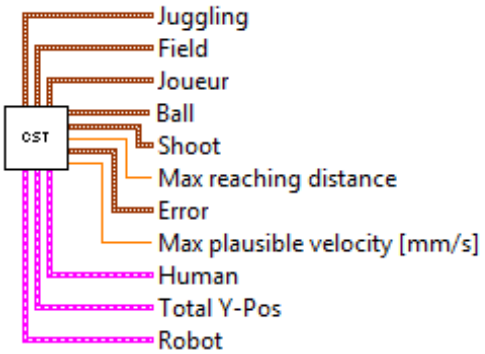


Figure 5.5: Constant

6 Conclusion

This semester project shows a new ways to deal with the strategies implementations and has great properties for future work. With this new way of programming, strategy can be changed just by changing the strategy path. Which mean that an infinite configuration can be stored without altering the code. Future work can be now done with the same main code for execution ! No need to copy past the project. For example, if we want to improve the pass command, we can just make a strategy with only the pass and work on the VI pass. Same with everything else. Since everything is now separated, multiple action can be improved in the same time without interference. And this is the strength of this project.

Objectives of this work has been mostly achieved. The main code seems to work quite well. Some adjustment of how the actions should occur still need to be done but from current testing, most problem come from imprecision in player positioning and ball velocity. Some work must be done to improve that. This work also revealed a problem in player shifting (for example when the forward right player shift with the forward middle player). The current configuration makes weak spot where the ball can easily go past the defense.

A Annex

0	$\notin$	Not in the range of [value1; value2]
1	$\in$	In the range of [value1; value2]
2	$\leq$	Less or equal to value1
3	$\geq$	Greater or equal to value1
4	$\neq$	Not equal to value1
5	$\equiv$	Equal to value1

Table 5: Inequalities indexes