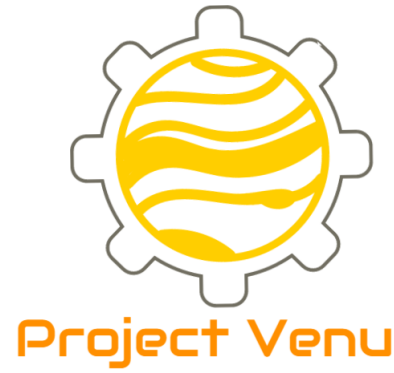


Technical Specification

SDP Group 15-H

Julijonas Kikutis
Patrick Green
Aseem Narang
Ankit Sonkar
David McArthur
Emilia Bogdanova

University of Edinburgh

Contents

1 System architecture

Venus has gone through several stages of changes. The earliest prototype had four regular wheels as seen in Figure 1. This made us realise that we have to keep in mind the space constraints when building the base as we need to accommodate not only the motors for the wheels but also the kicking and grabbing mechanism.
The next, more sophisticated design, was a robot with four powerful kickers that used elastics as seen in Figure 2. The idea was to stretch elastics for all four kickers using the star shape element on top of the robot as seen in the picture and then release. This gave a lot of energy to the ball. However, this design had a disadvantage because the robot was not able to kick the ball for varying distances, which was an essential requirement for the first milestone. Moreover, the dimensions of the robot were exceeding the maximum size limits and the speed of the ball was exceeding the one allowed in the game rules.

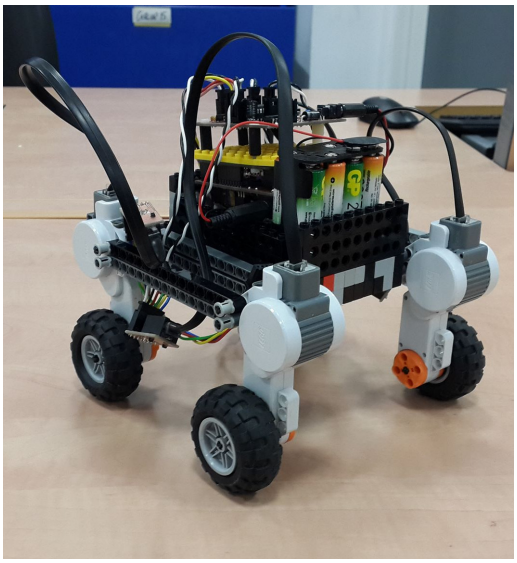


Figure 1: The first prototype

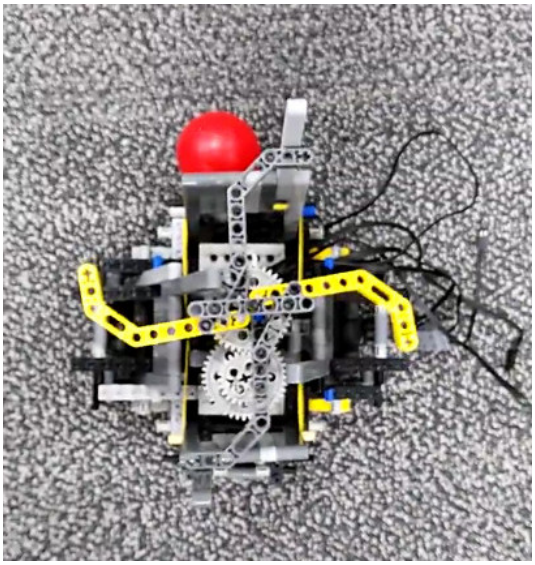


Figure 2: The robot with four kickers

Taking all the disadvantages of previous designs into account we came up with a new design as seen in Figures 4 and 3. Advantage of this design is that it is simple and at the same time the robot can still move fast enough to play football and the power of the kicks can be precisely controlled.

The grabbing mechanism for this design went through two iterations of development. The first grabber we developed was falling down on the ball in order to catch it, thus was deemed not suitable because it obstructed the ball more than it was allowed by the football game rules. Venus also had difficulties at placing the ball in front of the kicker at initial kicking position, because the grabber pushed it too far inside the kicker. Thus, the grabber was replaced by the newer design depicted in Figure 4 which has two symmetrical grabbers that interlock when the robot is moving to conserve space and are able to reach a more distant ball.

After playing the first friendly, we have realized that Venus isn't fast enough comparing to other robots. We decided to start the design from scratch again. Our latest design has four holonomic wheels, which allow Venus to move fast and also in different directions (Figure FIGURE). We had also refused of using any kickers, because none of the kickers we thought

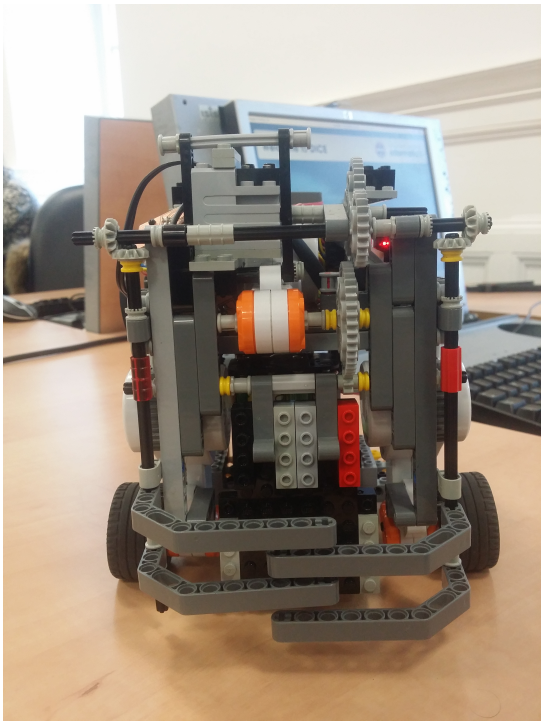


Figure 3: With grabber in closed position

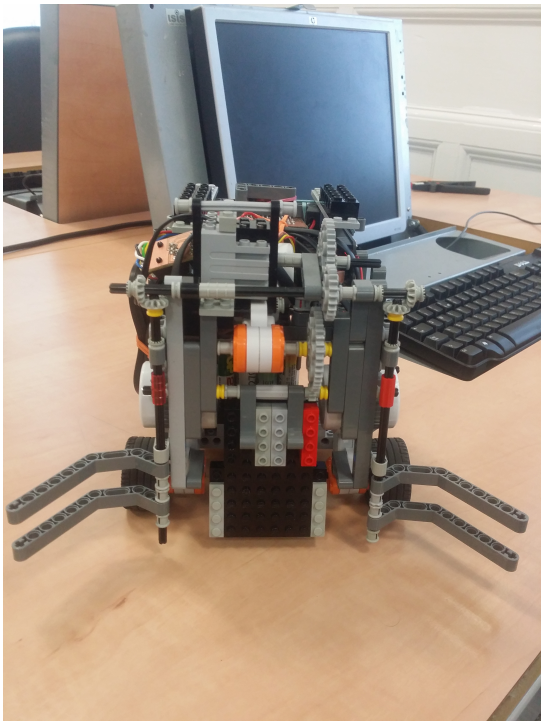


Figure 4: With grabber in open position

of, could fit with four NXT motors. Venus kicks using its grabber and spinning action. The design of the grabber hasn't changed much from the previous time.

When we were satisfied with the look of Venus, we also made all cables look neat using elastics. We also ensured that we can still access the battery pack and Arduino board if needed as seen in Figure 5.

1.1 Old Kicker

Even though the idea of the kicker that uses elastics was appealing as this made the kick more powerful, it was almost impossible to predict the distance the ball will travel. Because of that the newer design had a kicker that was powered by an NXT motor with gears. The kicker operated by going backwards inside the robot from the starting low position as seen in Figures **??-??** and then kicked forwards and touched the ball.

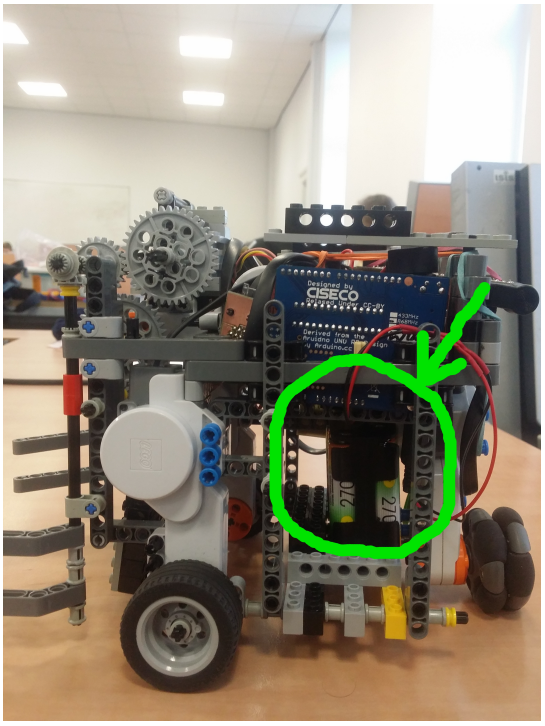


Figure 5: Battery pack easily accessible

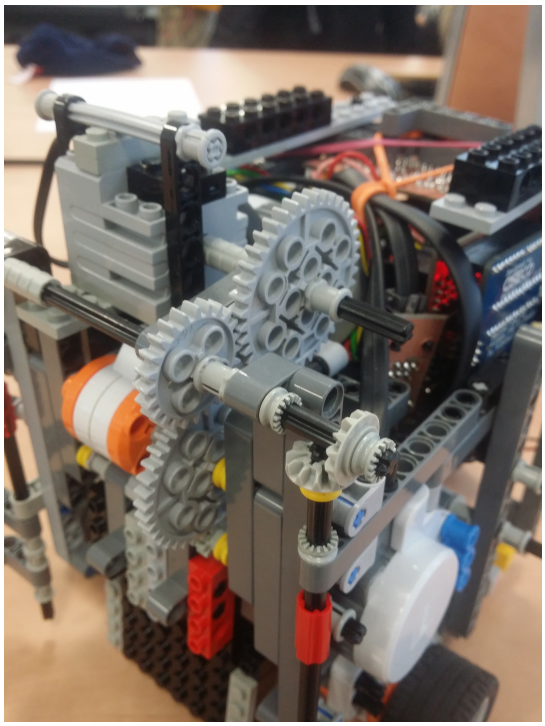
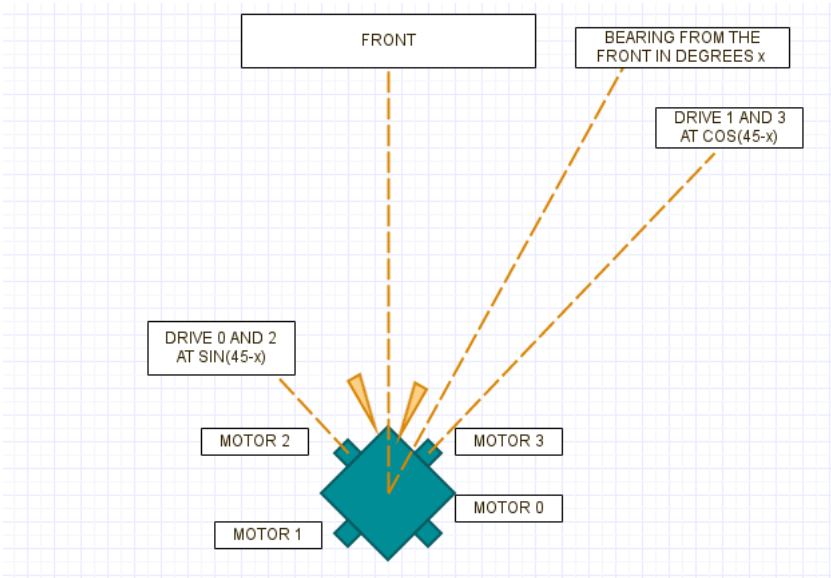


Figure 6: Gears for the grabber

2 Hardware components

2.1 Wheels



Holonomic motion was an obvious choice for our final robot. After calculating a desired direction we used the angle between that an the orientation to compute the driving components we needed to send to the Arduino, this is visible in the image above. The bellow matrix multiplication was used to extract the driving speeds and then each speed was scaled

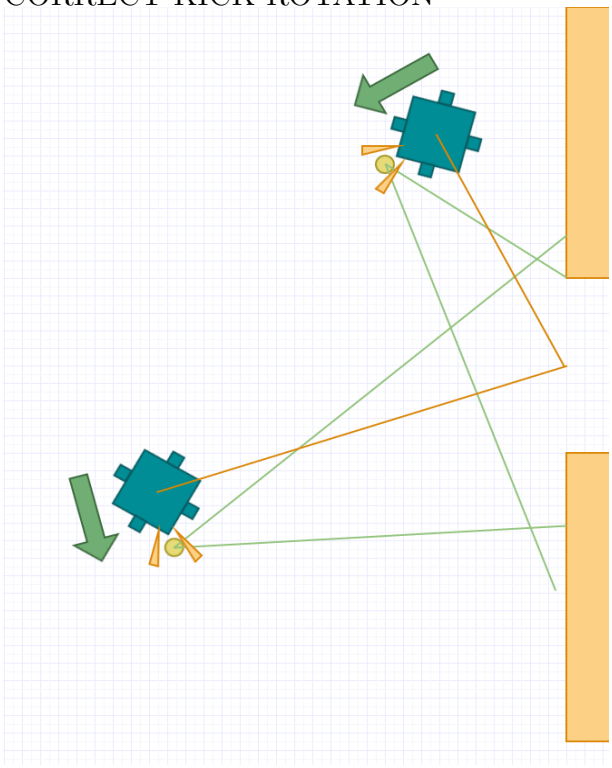
up by a factor of 100 divided by the absolute maximum of the 4 motors drive.

$$\begin{pmatrix} MOTOR & 0 \\ MOTOR & 1 \\ MOTOR & 2 \\ MOTOR & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \\ -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} cos(45-x) \\ sin(45-x) \end{pmatrix}$$

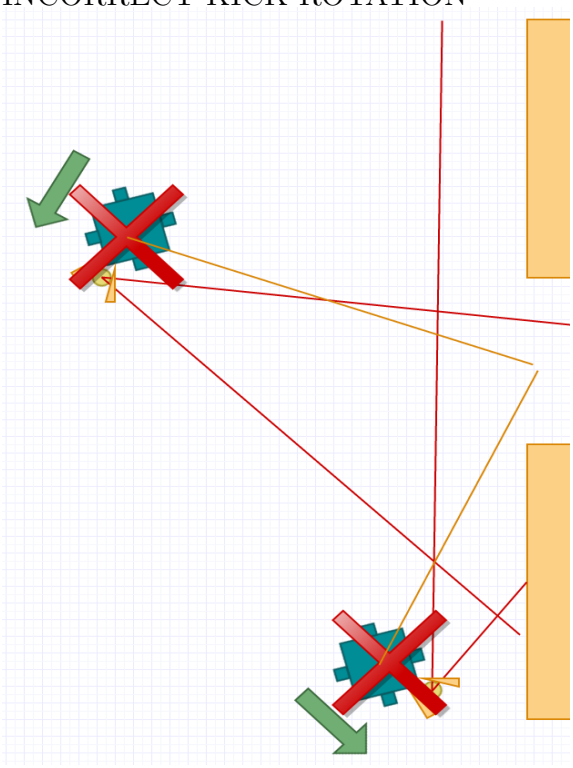
2.2 Kicker

The Newest Design

CORRECT KICK ROTATION



INCORRECT KICK ROTATION



To obtain consist kicks when setting up the angle for the kick the Venus is always rotated in the kick direction such that the centrifugal force holds the ball towards the end of the grabber and touching the claw that the ball will roll down and out of when the kick is initiated. As the error margin of the kick is asymmetric and biased towards overshooting the kick is initiated in different directions in different positions on the pitch. The y dimension is split into 4 segments $\frac{0}{4}$, $\frac{1}{4}$, $\frac{2}{4}$, $\frac{3}{4}$. In the diagram above each robot is demonstrating each segment.

2.3 Grabber

IS THIS NECESSARY ? The grabber for the previous design consisted of two symmetric parts placed one a bit above the other and was placed on the sides of the front part of the robot (Figure 6). They interlocked when the robot needs to catch the ball and we also kept them in a closed position during movements without the ball. We opened the grabber only before catching. This ensured that the grabber don't get broken against other robots or walls. We have used two bevel gears in order to be able to move grabbers with motor being

on top of the robot. The current design of the grabber doesn't differ much and can be seen in Figure FIGURE. The grabber is powered by a single Electric Technic Mini-Motor 9v with gears.

3 Documentation of the code

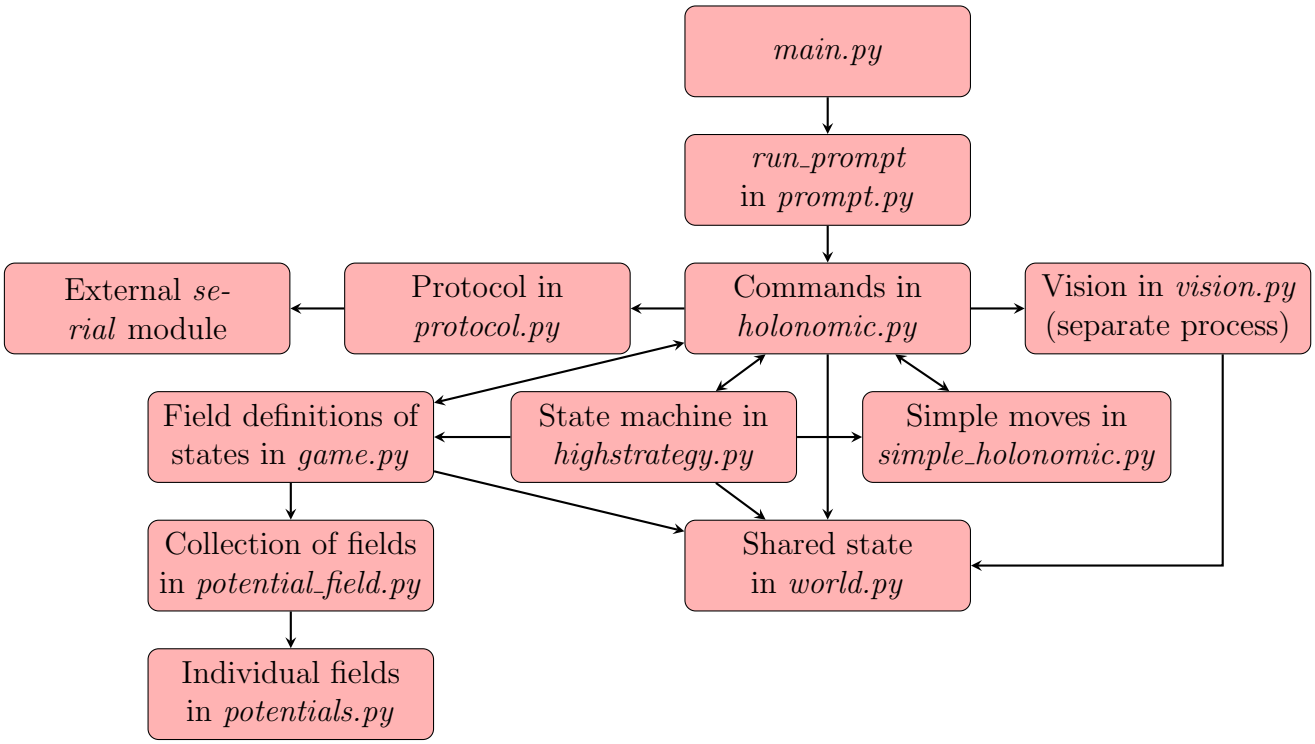


Figure 7: Dependency graph of Python modules in the system

The code is subdivided into several Python modules as seen in Figure 7. The module *main* is used to launch the *prompt* module. The commands in the prompt are provided by the *commands* module. It creates instances of protocol, vision, and state machine objects from the respective modules. The vision object is run on a separate thread to process frames asynchronously with constantly updated shared world state kept in a *World* object. At the same time the prompt is provided for the user. The command to run the strategy is called `hs`. This command constantly queries the state machine in the *highstrategy* module. The state machine checks the world state to decide which state it is currently in and then performs the associated action. It is done either by handing over the execution to *game* module to construct a potential field and perform the best action based on it or performing a predefined move from the *simple_holonomic* module.

All code except the colour calibration user interface in the *vision* module is an original work. Libraries used are *pyserial*, *numpy*, *scipy*, and *OpenCV*.

3.1 Communications

The communication interface between the Arduino and PC is low level as the PC decides and specifies the individual motor numbers and rotary encoder value or time duration for

which they will be powered. Then the Arduino sends acknowledgement to the PC about the arrival of the command, turns the motors on, and sets the specified timeouts to stop them. The messages are human-readable, newline-terminated and tokens inside them are separated by spaces. The specified motor power can be negative, in which case it means backwards direction. Each message which changes the state of the robot has a sequence number and checksum that are checked in the Arduino. The messages used in the protocol are listed in Table 1. The communication messages are constructed in the *Protocol* class using methods named after the corresponding message in `control/protocol.py` and these are used in the motion commands in `control/holonomic.py`.

Rotate the motors for n ms	`M seqNo checksum n motorNo power...`
Rotate the motors for n rotary values	`R seqNo checksum n motorNo power...`
Rotate the motors indefinitely	`V seqNo checksum motorNo power...`
Stop all motors	`S seqNo checksum`
Return `D` if all motors are stopped	`I`
Handshake, reset the sequence number	`H`
Query light sensor, return `D` or `N`	`A threshold`
Transfer a byte to I2C bus	`T byteInDecimalASCII`

Table 1: Messages available in the protocol

3.2 Arduino

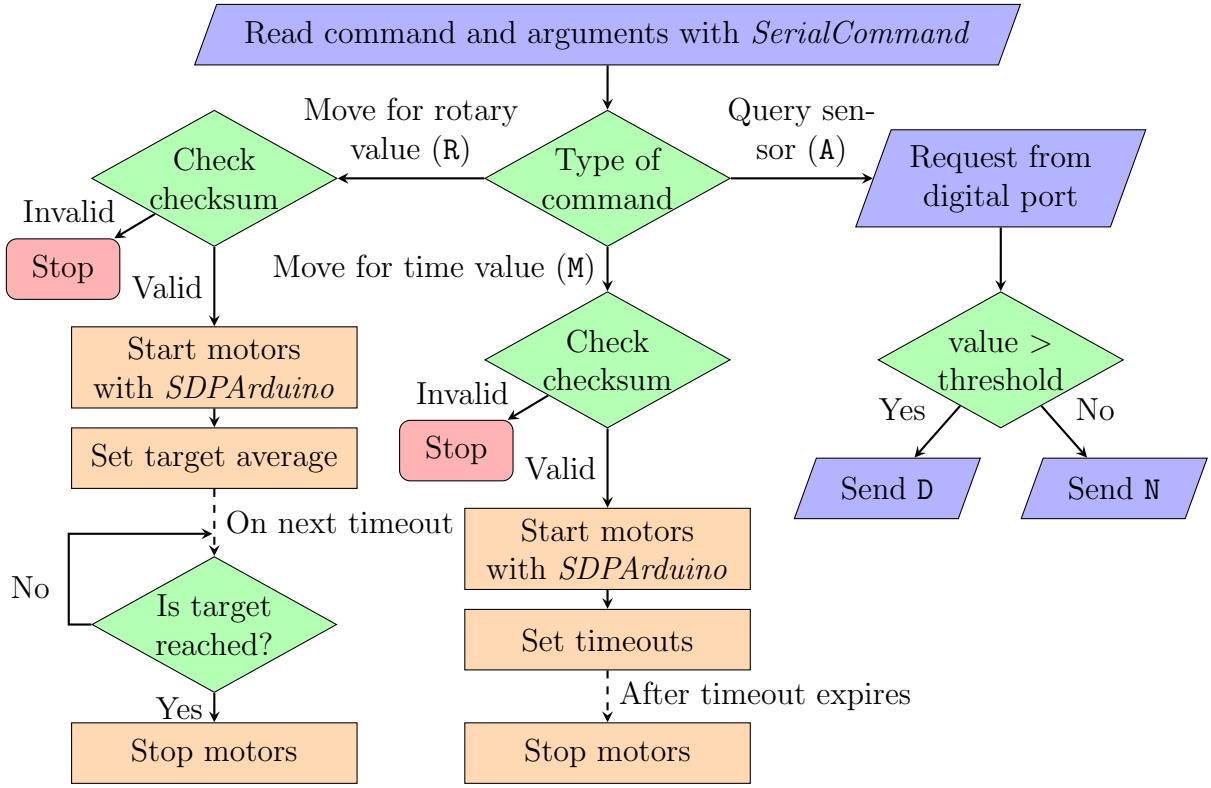


Figure 8: Flowchart of message processing in the Arduino

The Arduino code uses the *SerialCommand* library to buffer and tokenize the commands received over the serial link. Messages changing the world state are sent with a sequence

number and checksum which is the sum of all the parameters following it. If the sequence number matches the one from the last command, acknowledgement is sent but no further work is performed for the duplicate message. This handles the situation arising when an acknowledgement from the Arduino does not reach the PC and and the PC generates a duplicate message. If the checksum does not match, the Arduino code ignores the message and does not send an acknowledgement, thus forcing the PC to send the same message again.

If any of the motor move commands are sent, the motors are started immediately using the *SDPArduino* library. Then the Arduino schedules when to stop the motors and there are two methods to perform that: either a time value or rotary encoder value. The flowchart for these two methods, along with querying the light sensor, are detailed in Figure 8. In the case of the time value, a timeout is set to stop each single motor using *setTimeout* from the *SimpleTimer* library. In the case of rotary encoder value, *setInterval* is used which calls a function every 5 ms that queries the rotary encoder board and stops the motors when the average of all four motor rotary encoder values reaches the target value. These approaches using timers allows the robot to receive commands asynchronously, that is, a command is not blocking during its execution and the PC software could, for example, send another command simultaneously to engage the kicker while the robot is in motion. The Arduino message handling is located in `arduino/arduino.ino` file.

A buffer of upcoming motor jobs has also been implemented in the Arduino code to ensure continuous motion but it was deemed unnecessary as vision could issue new commands quickly enough without the robot coming to halt. The current implementation also never stops a motor when a new command arrives for the same motor, instead it just executes the new command, ensuring continuous motion.

3.3 Vision

In our opinion, the vision is our unique feature of this project. We have built the whole system from scratch without borrowing any code from previous years. This allowed us to implement things we might not have been able to do with an existing system.
In order to be able to detect robots and the ball, we go through the following steps:

1. The dictionary of values considered in the camera settings (Brightness, Contrast, Hue and Saturation) are read from room?.txt, pitch dimensions read from pitch?.txt and color thresholds are read from color?.txt.

2. The camera is capped by the previous settings and a frame is read and Gaussian blurred to create more solid features.

3. A unique application to speed up the feed is the variable world.undistort which lets the strategy decide whether to remove the barrel distortion or not.

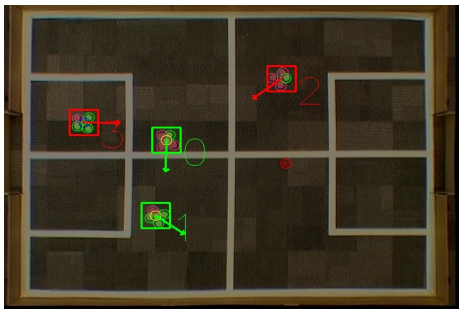
4. All color thresholds are added to the same mask and kmeans is used to group together the various spots in the image.

5. The color is then found from the center pixel of the cluster and the spot is either kept or removed depending on the minimum area for that color which varies as some colors are harder to see.

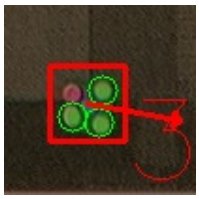
6. The individual methods required for finding the robots and ball are then implemented and outline bellow.
7. When the vision system is exited through the escape key the most recent calibration values are saved and can be used on the next run and the current camera settings defined by the slider bars are saved also.

3.3.1 Finding the robots

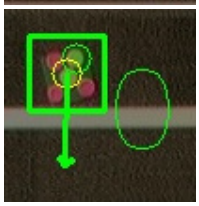
Due to the varying light intensities over the pitch we built a very mailable method of capturing robots. The robot id's are pre-defined and dependent on our choice of center and corner spot.



Firstly there are 2 independent ways a robots data can be captured

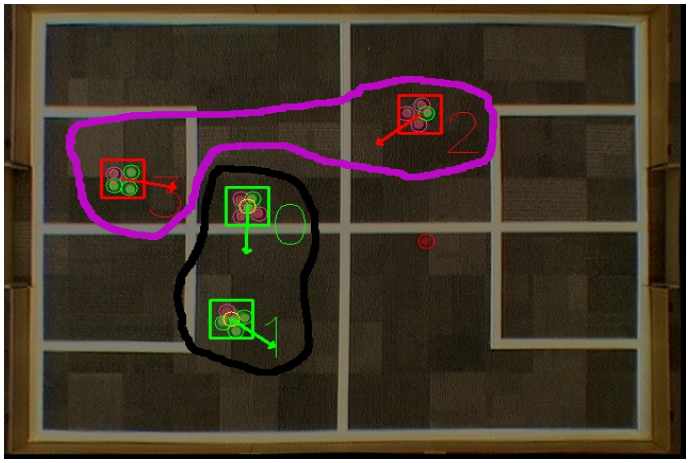


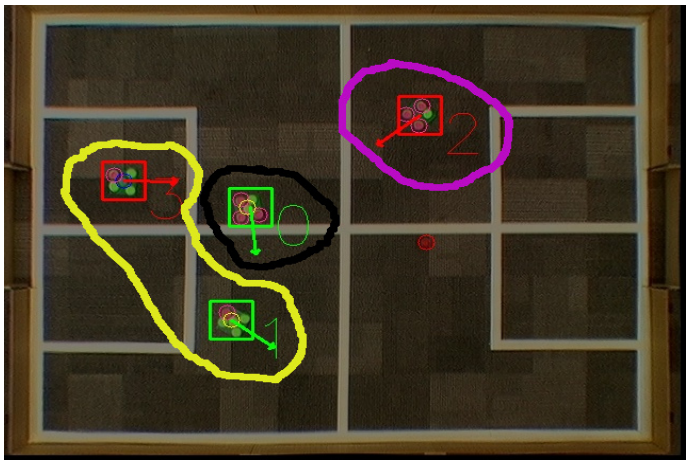
THE 3 SPOT METHOD: finds the largest of three distances between spots, it then draws a vector from the mid point of this distance to the spot not included in the maximum distance. The orientation is then computable by rotating this vector a fixed amount. The robot center is calculated using the average of the spot centres.



THE 2 SPOT METHOD: As each spot is distinguishable, it can therefore construct a vector between the team spot (center spot) and the corner spot and rotate it a fixed amount to find the orientation. The robot center is calculated using the team spot center.

As there will always be 2 robots with the same configurations required for the methods above the algorithm requires some robot id's to be found with more information. As you can see above robots on identical teams will have the same two spot configuration and for the three spot method the robots 0,2 and 1,3 will have the same configuration. To solve this we firstly group the spots together into potential robots in order of descending areas. We the run the groups through several filters for each robot id accepting the groups if there exists a specific numbers of coloured spots. Each spot that is found is circled in its respective color on the vision feed.





FILTER 1 To pass into here it must have a correct team spot and no other visible team color. It must also have more than one of its three spot color.

FILTER 2 To pass into here it must have a correct team spot. It also must have more than one of its three spot color or less than 2 of its corner spot color.

FILTER 3 To pass into here no team spot is required , it must only have more than one of its three spot color or less than 2 of its corner spot color.

As each id is found its corresponding filters are blocked and its position is noted so that a robot is not found in the same place of one that is already found. Once any group of spots gets through a filter of a given id, the id of that robot is then updated using either the 3 spot or 2 spot method depending on what information you have. If any robot is not found it keeps its original position. To handle cases where robots are lifted off the pitch the system creates a ghost robot on the vision marked in blue. This tells the strategy to remove that robot id's contribution to the game but in the vision thread the robot continues existing on the pitch.

3.3.2 Finding the ball

The algorithm looks for 10 red spots and checks through them in order of descending area. To prevent a pink being misclassified as the ball each red spot is checked to see if it exist close to the robot centres. As the robots can shield the ball from view we include a method which determines whether a specific robot is in range of the ball. If so, when the ball is shielded an imaginary ball is placed along that robots orientation vector until it is found. This method is in place for all robots but Venus as we use our sensor to determine whether we have it.

3.4 Strategy

3.4.1 Potential Fields

Our strategy relies on the physical properties of a potential field. This meant we were able to compound together separate tasks that the robot was to undertake with ease, such as avoiding others and grabbing a ball. We implemented each interaction with a physical objects as a charged electron would interact with varies charged objects. Through this we were able to to swerve in front kicks and navigate mazes of objects fast and efficiently.

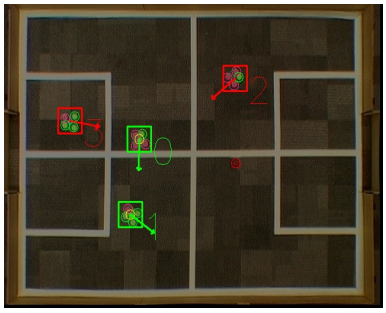
3.4.2 State Machine

State	Trigger
$FREE\_BALL\_YOURS$	
$ENEMY\_BALL\_TAKE\_PASS$	
$ENEMY1\_BALL\_TAKE\_GOAL$	
$ENEMY2\_BALL\_TAKE\_GOAL$	
$FREE\_BALL\_NONE\_GOALSIDE$	
$FREE\_BALL\_1\_GOALSIDE$	
$FREE\_BALL\_2\_GOALSIDE$	
$FREE\_BALL\_BOTH\_GOALSIDE$	
$ATTACK\_PASS$	
$ATTACK\_GOAL$	
$RECIEVE\_PASS$	

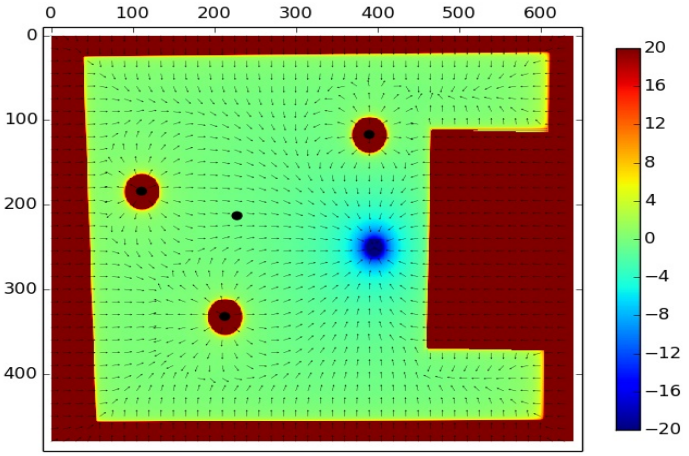
3.4.3 State Fields

The graphs bellow contain a quiver plot of vectors in the direction of forces in a grid of points on top of a heat map of the fields potential at a given point. It is plotted using the command '$map \quad STATE\_NAME$' and is ideal for fast testing of new fields.

$FREE\_BALL\_YOURS$

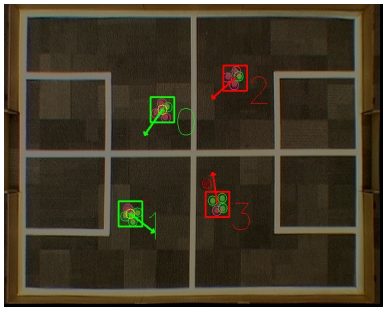


FORCES DUE TO OBSTACLES: The walls use a $1/r^3$ law on the pitch side of the wall where r is the perpendicular distance to the wall. On the opposite side of the wall we switch to an attractive potential to pull any robots into the pitch $-1/r^3$. An identical field is used in the penalty box however to avoid getting stuck in local minima at the sides the field inside the box pushes the robot to the front and doesn't attract it back over the side it came. As the box is finite any position not perpendicular will exhibit a similar force only r will be the distance to the closest corner. The robots use a $1/r^2$ where are r is the radial distance from the edge of the robot (represented by the solid circle).



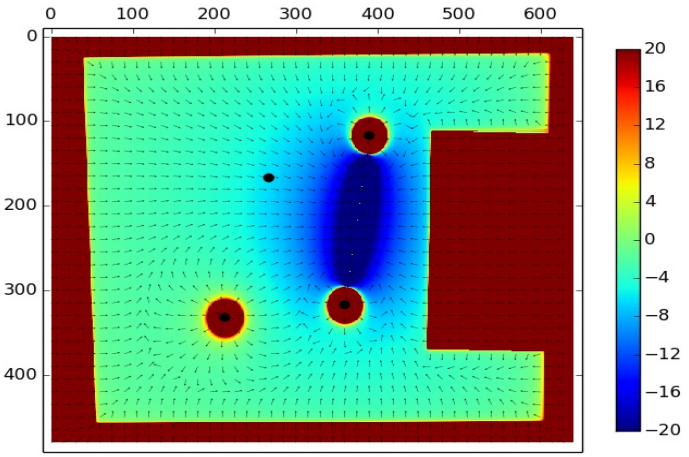
GRABBING: The ball uses an attractive radial field of $1/r^2$. The amplitude of the field is larger than that for the robot to enable fast navigation. Once a potential of -4 is reached we would initiate the grab using specific distance and angle motions using the motor encoders. -4 was chosen as it implied the ball was clear of any obstacles as the potential value would be higher.

ENEMY_BALL_TAKE_PASS



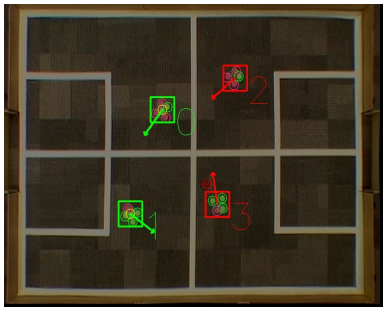
We use a finite axial field to enable smooth motion into the block of the pass from any point in the pitch. the points in question are first rotated so the field is flush with the x axis and the force is calculated using the following equation and rotated back. d is the perpendicular distance, a is the distance to the end with a smaller x value and b the exact opposite.

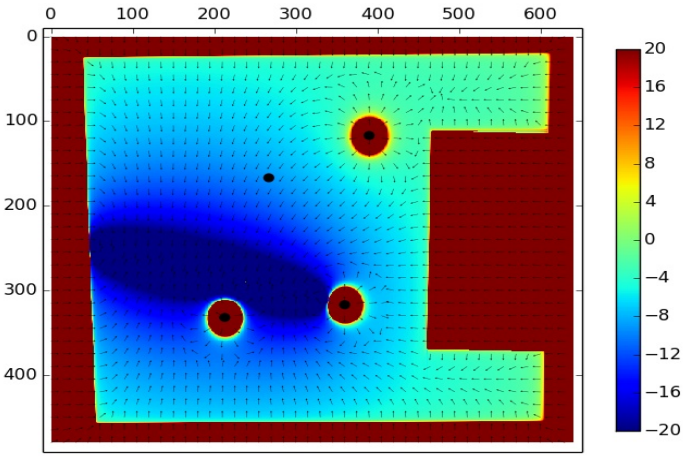
$$\left(\frac{1}{\sqrt{b^2+d^2}}-\frac{1}{\sqrt{a^2+d^2}},\frac{b}{d\sqrt{b^2+d^2}}+\frac{a}{d\sqrt{a^2+d^2}}\right)$$



When Venus's current potential reaches -12 the block is satisfactory and it faces the ball with grabber open. When the ball is shot as it is moving FREE_BALL_YOURS will be activated saving the shot. Only when the ball stops, the grab be initiated.

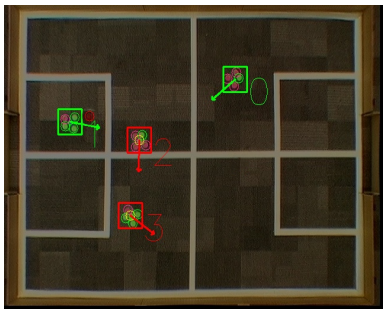
ENEMY2_BALL_TAKE_GOAL (*ENEMY1_BALL_TAKE_GOAL*)

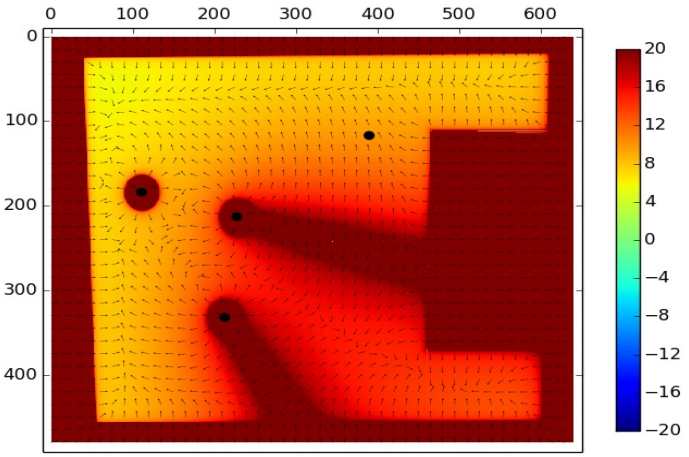




The same field is used from the state above but for a block between the goal and an enemy robot. The satisfactory potential for this shot is -14 in order to be sure of blocking the whole goal.

FREE_BALL_NONE_GOALSIDE (*FREE_BALL_1_GOALSIDE*,*FREE_BALL_2_GOALSIDE*)





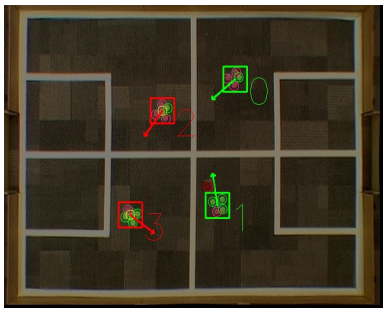
Given you team mate is either fetching or has the ball you want to find the best position to receive a pass. We add 2 fields in order to implement this:

1. the shadowed array blocking the pass
2. the shadowed array blocking a goal

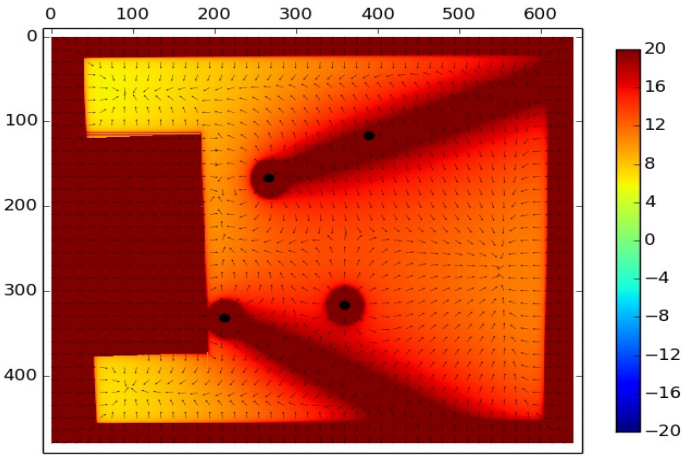
This image is representing a double blocked pass. We use an identical field to that in the defence states however we place one end at the start of the shadow and the other 3 pitch lengths away. This is so Venus is repelled perpendicular to the shadow and also forward around the enemy robots if needs be.

Once a satisfactory potential of 6 is reached Venus will turn and wait to except the pass. As the other robots move Venus will keep trying to minimise his potential until a value of 6.

FREE_BALL_BOTH_GOALSIDE (*FREE_BALL_1_GOALSIDE*,*FREE_BALL_2_GOALSIDE*)



LOCAL MINIMA: In these states Venus can get stuck as enemy players close up to defend the same thing, to deal with this a timer is used so that if an its been sitting in a unsatisfactory potential we remove the least import field, in this case the furthest player blocking the goal.



As outlined above this graph represents a double blocked goal and the final 2 states not shown are a mixture of this and the one above.

---

4 Sensors

4.1 Rotary encoder board

Each NXT motor is connected to the rotary encoder board which is depicted in Figure 9. The connections in the anticlockwise order from the top are: the I2C bus to the Arduino, back right motor, back left motor, front left motor, and front right motor. Using this board the information about the amount of rotations the motor has performed since the last query is available for the Arduino code as a separate integer for every motor. Every 5 ms the board is queried whether the target value has been reached. After the average of the rotary values of all four motors becomes greater or equal to the target value, the motors are stopped.

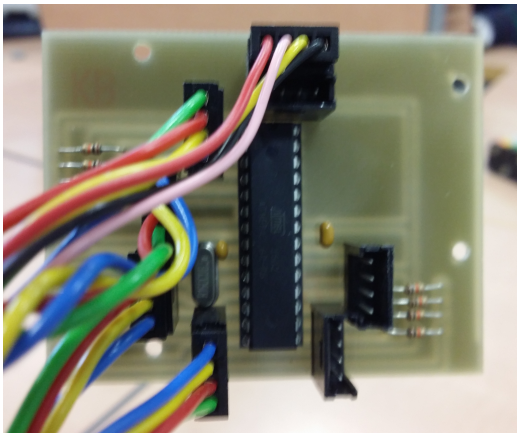


Figure 9: The rotary encoder board

4.2 Light sensor

The light sensor is located above the grabber as seen in Figure 10. It is used to check whether the robot has successfully acquired the ball after grabbing. The sensor provides an integer value that increases the greater the distance to the nearest object in its direct line of sight is. Then the Arduino compares the integer to a predefined threshold corresponding to the red ball. Sometimes a white line inside the pitch can be mistaken for the ball due to their similar sensor values. An IR sensor was also tested and its returned values were deemed less reliable than those of the light sensor.

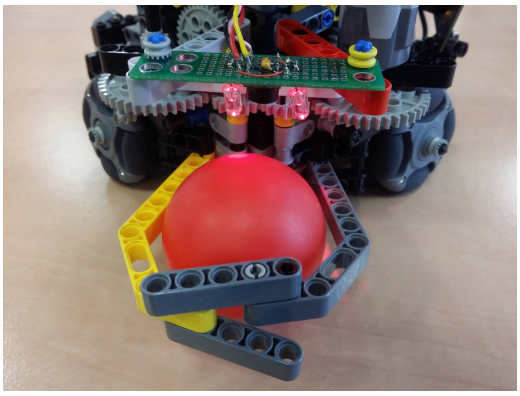


Figure 10: The light sensor with the ball grabbed