



Project Venus

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	2
2	Hardware components	3
2.1	Wheels	3
2.2	Kicker	3
2.3	Grabber	4
3	Documentation of the code	5
3.1	Communications	5
3.2	Arduino	5
3.3	Vision	8
3.3.1	Calibration	8
3.3.2	Finding the robots	8
3.3.3	Finding the ball	9
3.4	Strategy	9
4	Sensors	9
4.1	Rotary encoder board	9
4.2	Light sensor	10

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.

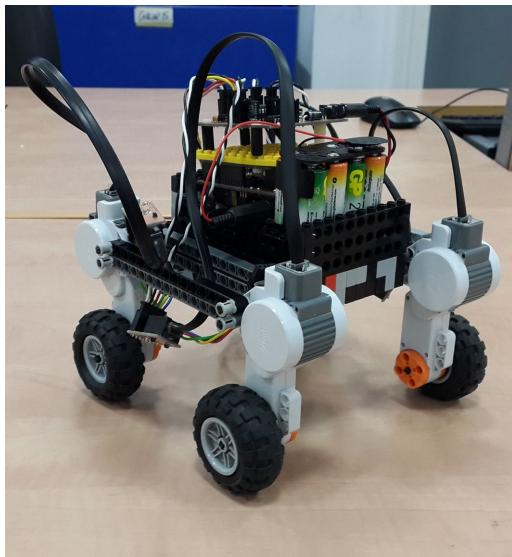


Figure 1: The first prototype

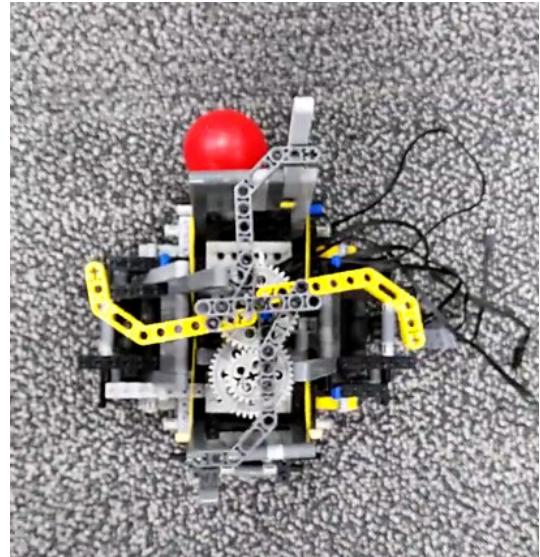


Figure 2: The robot with four kickers

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

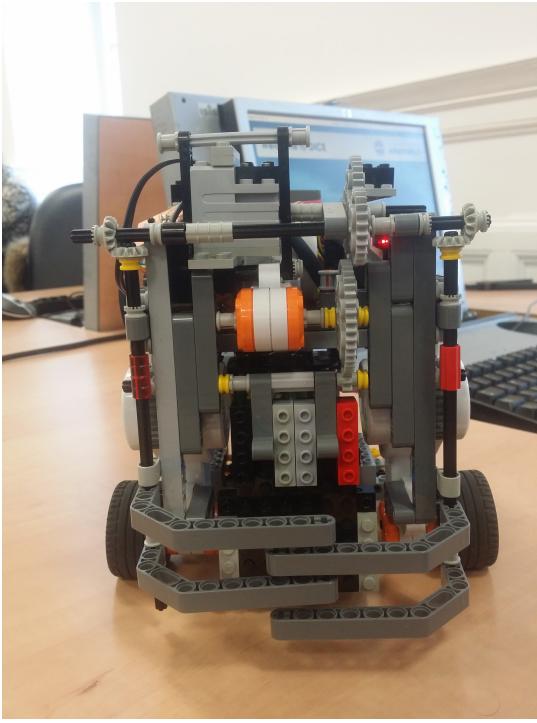


Figure 3: With grabber in closed position

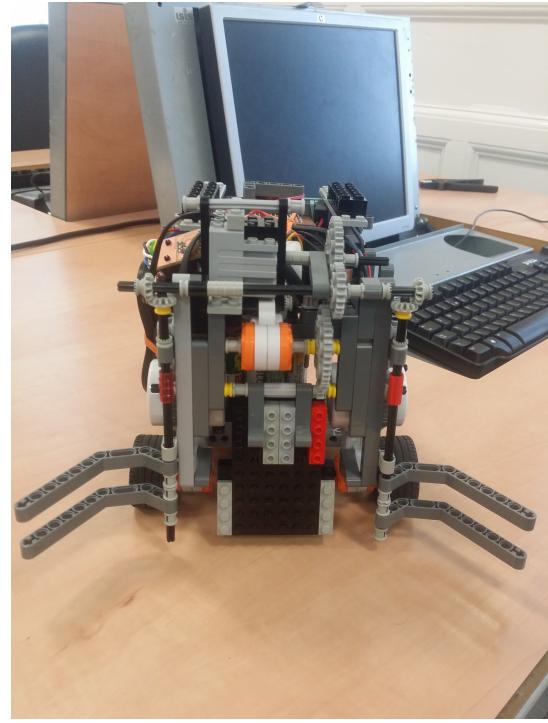


Figure 4: With grabber in open position

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.

2 Hardware components

2.1 Wheels

Could you be more descriptive? i.e. why you decided to build it like that ? Why mixture of wheel types ? How the robot can turn? what kind of problems you were trying to omit ? It is a far to short and vague description. Since your robot has four holonomic wheels now, could you describe why it is a four-wheeled robot now? What is the main feature, advantage of the holonomic property? What is exactly this property. What are NXT motors ? how they work?

The current design has four holonomic wheels, which allows Venus to move in any direction. All wheels are powered by NXT motors.

2.2 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



Figure 5: Battery pack easily accessible

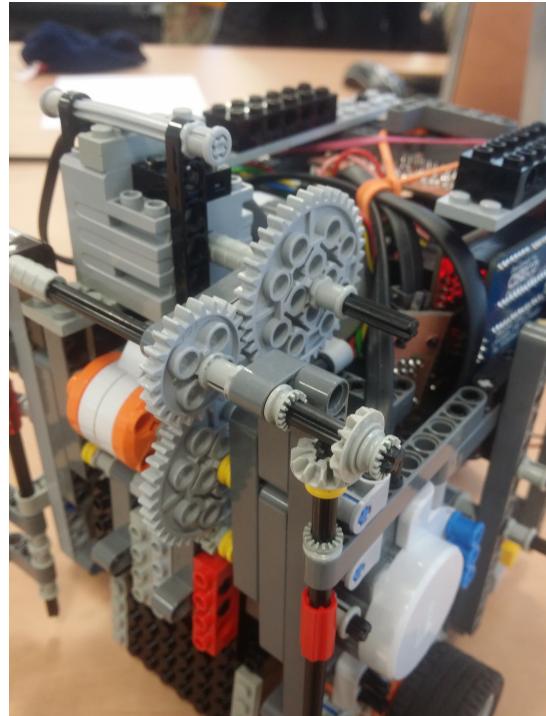


Figure 6: Gears for the grabber

kicker operated by going backwards inside the robot from the starting low position as seen in Figures 7-9 and then kicked forwards and touched the ball.

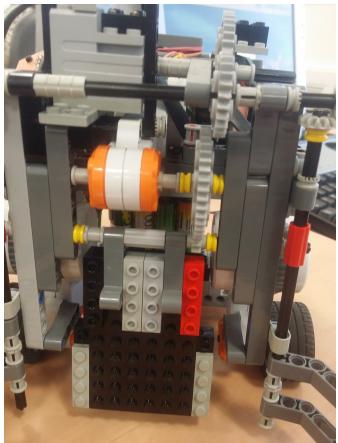


Figure 7: Kicker in still position



Figure 8: Kicker starts going back

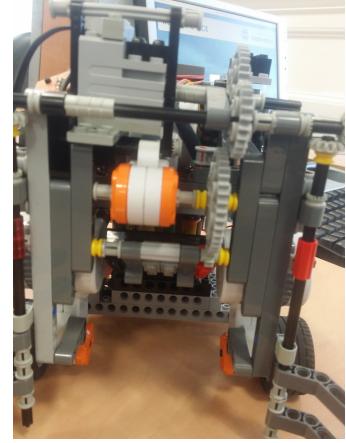


Figure 9: Kicker in the back position

Our newest design doesn't have a kicker. The kick is made by turning the robot and opening the grabber at the same time. This creates a momentum necessary to make a kick (Figure 10).

2.3 Grabber

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).

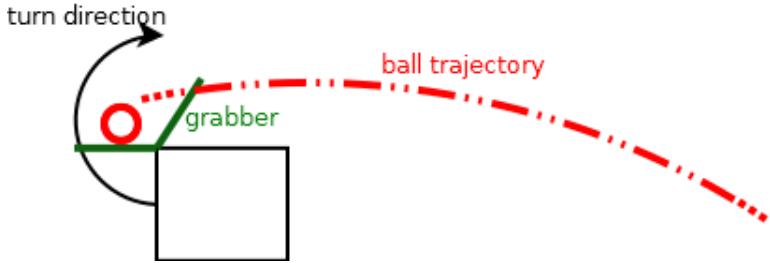


Figure 10: High level diagram of how the kick is performed

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

The high-level overview of the system structure can be found in Figure 11. It is not fully implemented yet.

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 `Protocol` class in `protocol.py` and used in motion functions in `holonomic.py`. The PC currently provides a simple command line interface. We use commands specified in the user guide such as `f 100` meaning 'go forwards 100 meters' that are mapped to communication messages, in our python code. We then are able to run this code from the command line.

3.2 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

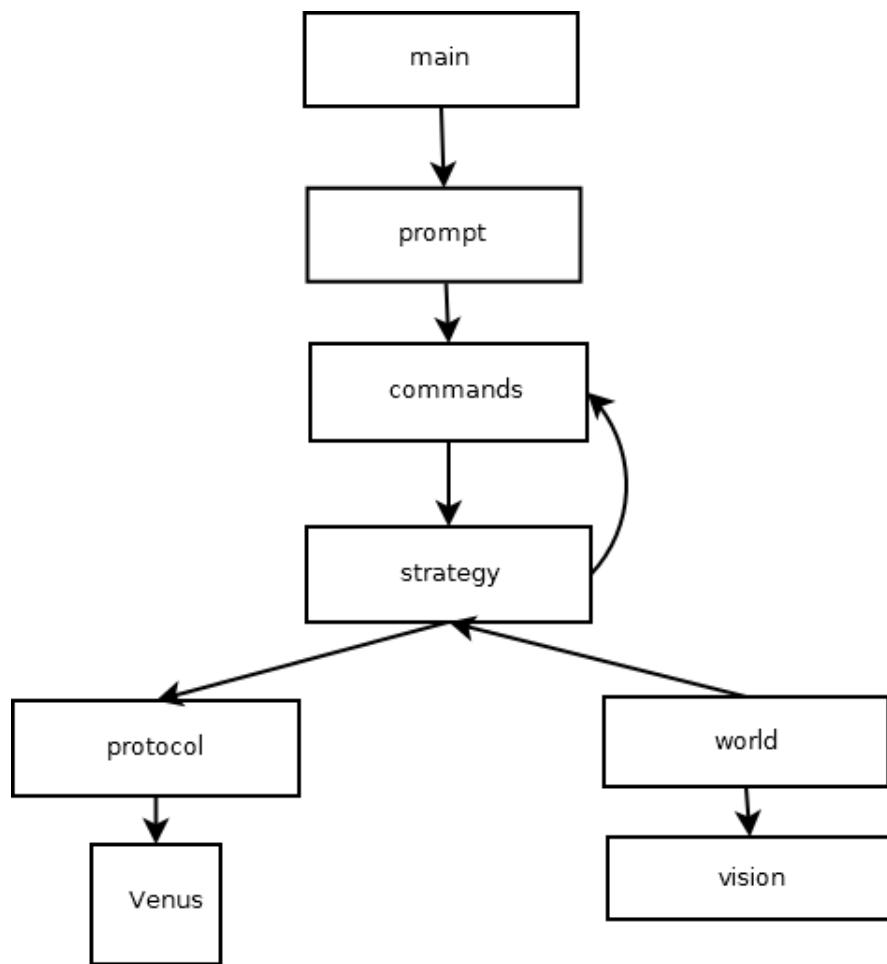


Figure 11: High level diagram of the system structure

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

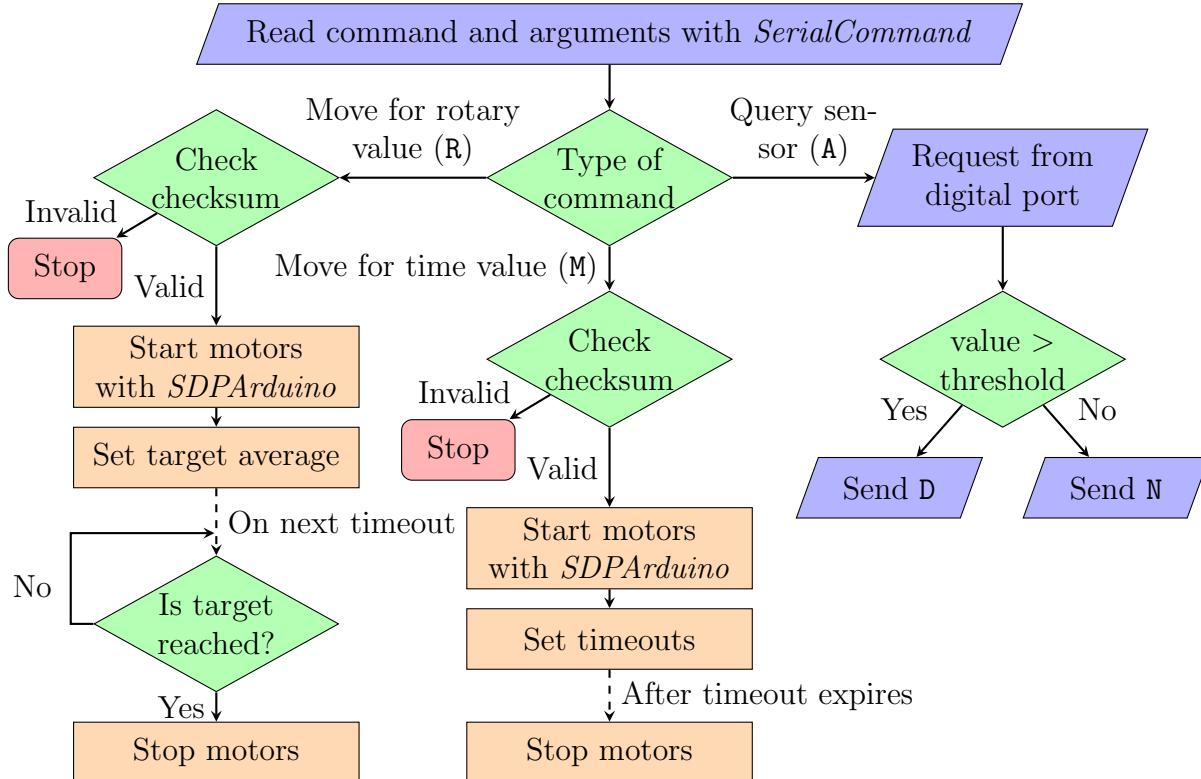


Figure 12: Flowchart of message processing in the Arduino

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 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.

In case of the motor move command, 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 12. 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 message handling is defined in `arduino/arduino.ino` file.

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. For the relevant room (0 or 1) read files containing the following data: brightness, contrast, saturation, hue values (room0.txt), minimum and maximum HSV values for each color that we want to detect (color0.txt), distortion (dist0.txt).
2. The image from the video feed is blurred (using `cv2.GaussianBlur`) and the mask is created (there is a possibility of calibration if some colors are not detected well enough).
3. We draw circles of relevant colors that are in the mask except red (the ball).
4. Find the ball.
5. Find robots.

3.3.1 Calibration

When the video feed is opened, there are bars on the top that allow you to adjust brightness, hue, saturation and contrast. There is also a bar that is called 'calibration', which takes boolean values 0 or 1. If you want to calibrate, you just need to move this bar to the right. When calibration mode is chosen another frame is opened, (it is a picture of the pitch captured when you entered this mode). In the top left corner, there is a ball representing the color that is currently being calibrated. In order to calibrate, you need to click on the pixels of this color. If the values that are read when you click do not match those expected for this color, they are discarded. With each new click a HSV values are stored, when calibration is finished, the range of all clicks is calculated for each color. This means you need to click multiple times in order to get a better result. This range is then stored to a file `color0.txt` overwriting anything else that was there before. We have minimum and maximum values for all colors except red, which has four values, because of the HSV cone. If you want to switch color, press 'q'. After all colors are done, the calibration mode is exited automatically.

3.3.2 Finding the robots

Once we have detected all the colors, we try to decide where the actual robots are. In order to do that we need all robots to be on the pitch in the beginning when we run the code. There is a list of four robots we need to detect. We sort all colors we detected by their size (from biggest to lowest). Then for each robot, we check the following:

1. If we see all the colors on approximately right distances (given in rules) from each other, it is a perfect case and we say we have detected the robot.

2. If we can see the centre and the spot that is in bottom left corner (the one that differs from others), then we know the centre of mass, and we draw an orientation vector by rotating the line between centre and the spot we detected.
3. If we only can detect a centre spot, then we draw an orientation vector that we have seen in previous frame.
4. If there are three spots of the same color, then the centre of mass is calculated by taking the centre of two furthest from each other points. It is not exact, but gives a good approximation. The orientation vector is then drawn through the centre of mass and the middle point of two front spots.

This ensures that we are able to detect robots even if at some point we cannot detect all the colors. Apart from the above steps, we also check that the centre of mass is on the reasonable distance from where it was in the last frame (to avoid robots flipping). Furthermore, if we have found the robot of one type (say our teammate), then we restrict it from finding another one.

When the robots are found we draw a box around them (green for Venus and yellow for teammate red for opposite team) and label with a number from 0 to 3, where 0 is Venus, 1 is our teammate and 3,4 are opposite team robots.

3.3.3 Finding the ball

We restrict the number of circles of each color we want to detect, e.g. 2 blue, 2 yellow, 8 green etc. For red, it is 9, because we are likely to think that pink is red. If at any point we are not able to see any red spots, we decide that the ball is underneath one of the robots or not on the pitch. If we can see the spot that is not part of the robots detected earlier, we store ball's position as a stack that has centres of mass of the ball for the last 6 frames (every time new value comes in, last one is popped). We check the first and the last values in this stack. If their difference is more than 10 pixels, then the ball is moving and we draw its trajectory.

3.4 Strategy

table with all states

4 Sensors

4.1 Rotary encoder board

Each NXT motor is connected to the rotary encoder board. This way the information about the amount of rotations the motor has performed since the last query is available for the Arduino code. 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.

4.2 Light sensor

The light sensor is located above the grabber. 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.