URoboRus 2019 Team Description Paper

Anastasiia Kornilova, Petr Konovalov, Galina Reneva, Dmitrii Iarosh, and Irina Khonakhbeeva

Saint Petersburg State University, Saint Petersburg, Russian Federation
`kornilova.anastasiia@gmail.com`

Abstract. URoboRus[1] is a team from the Saint Petersburg State University (Russia) developing a solution to participate in the RoboCup Soccer Small Size League. This paper presents the technical overview of our robots, control software system and main algorithms. It will be the first participation in this competition, consequently, a full description of all components is provided.

Keywords: RoboCup · robotics · multi-agent system · hybrid centralized system

1 Introduction

Our team unites passionate students from departments of Software Engineering and Theoretical Cybernetics (Control Theory), and evolves with a support from initiative faculty members of these departments. Also, our project is supported by CybetTech Labs Co Ltd, a company behind the educational robotics kit TRIK [1], which is widespread in Russian schools. We started to create our solution in September 2018 on the basis of robots, which were developed by third-party company on our request, and at the moment of writing we have implemented a framework for robots control and a library with general algorithms. By April-May 2019 we plan to support a full game by Robocup-SSL rules.

Successful participation in the professional Robocup-SSL league and consequent improvement of hardware and software are not the only aims for us. Related goals also include popularization of this competition in Russia, development of solution for educational robotics, which is affordable for schools and universities, and in the future – organization of local Robocup-SSL league in Russia. While reading this article one will see some solutions which have been done in this direction. For example, a porting of an SSL-Vision to Windows is in process, an attempt to make our control system cross-platform are being made too, furthermore opportunities for different interpretation engines (i.e., Python) are being considered. As of January 2019 we have already organized a hackathon for students at the Russian inter-university robotics school [2] and have participated with our setup in the international robotics festival ”Robofinist” [3].

[1] https://en.wikipedia.org/wiki/Uroborus

2 Robots description

As it was mentioned in introduction, robots, which we use, were made by third-party company by technical specification, so in the section we provide a description of this specification and some details about its implementation.

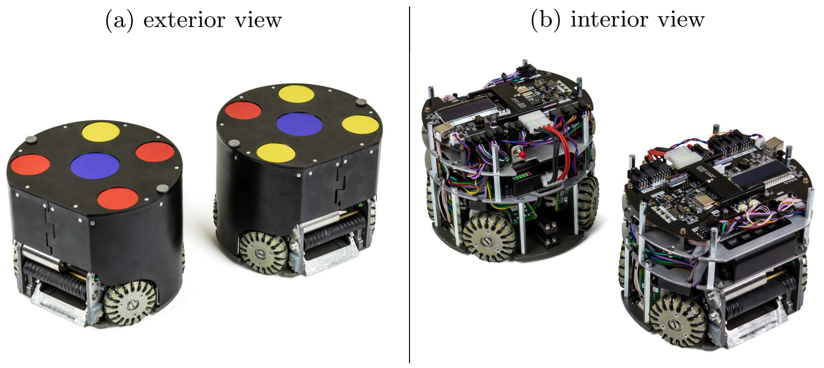


Fig. 1: Robots

The robot (Fig. 1) consists of a straight kicker, a chip kicker, four omnidirectional wheels, and a ball spinning device (dribbler), which is equipped with a ball presence sensor. The robot has inertial sensors and a Wi-Fi transceiver for communicating with the control software. An LCD screen is installed on the robot's motherboard and displays the name of the current Wi-Fi network and robot IP address in this network. The robot is equipped with a USB service interface to change internal parameters and restrictions. Each device is controlled by a separate board, that simplifies the process of diagnostics and repairing.

2.1 Main technical characteristics

- Diameter of the robot's body – 180mm, height – 146mm, weight – 3500g (Fig. 2)
- Maximal linear speed – 1.5 m/sec
- Maximal angular speed – 3 rev/sec
- Maximal kick speed – 3 m/sec, kick-up height – 250mm

Battery The robot is equipped with a removable battery with a capacity of 3000 mAh and an average voltage of 26 volts. The operation time is ≈30 minutes with an average current consumption of 6A. In order to charge the battery, it should be preliminarily removed out of the robot. The battery is equipped with a balancing device, as well as a protective device against overheating, short circuit, deep discharge and overcharge. It also has a charge level indicator on the front panel.

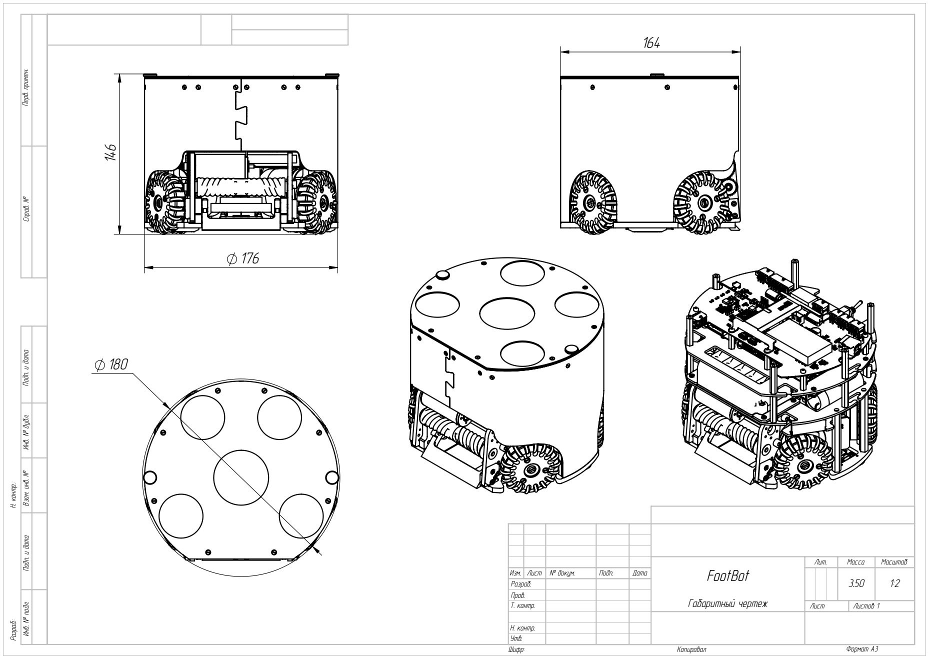


Fig. 2: Robot blueprint

Connection Connection with robot can be established via 2.4G Wi-Fi and it receives IP address automatically from DHCP. Protocol of communication with robots is defined in the section Robot communication module.

Motors The robot is equipped with four mid-flight brushless motors for omni-base and one more "dribbler" brushless motor to spin the ball. The motor control method is vector control with current control in the windings and limiting the maximum torque on the rotor. Every engine is served by a separate driver board with a microcontroller.

Robot control There is a special coordinate system associated with robot. (Fig. 3). So that the robot starts to perform actions, a special UDP packet with parameters should be sent to the robot. Movements of the robot are controlled by using SpeedX (speed along axe X), SpeedY (speed along axe Y), SpeedR (angular speed of the robot). To kick ball, paramaters Kick-up or Kick-forward should be set. For more details see in Robot communication module.

3 Control software

This software was developed by using C++ with Qt framework [4]; this choice was partly motivated by its cross-platform nature. Software implementation of

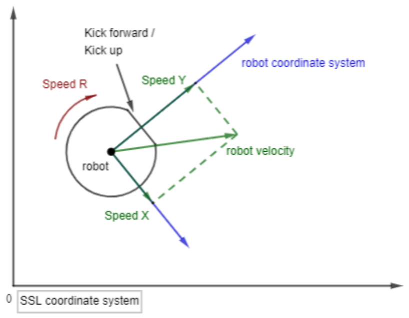


Fig. 3: Robot control

the algorithms was performed by using Matlab [5], partly due to its convenience at the stage of prototyping.

Our software system basically consists of the following two parts:

1. *Centralized control tool* [6] is commissioned to solve the following tasks:
   - collecting data about field geometry and game situation from robots and SSL Vision [7];
   - providing this data to the Matlab algorithm library, which calculates control signals for robots;
   - transmitting those signals to the robots.

   The connection with SSL server is established through SSL receiver module, which distributes the received data among all other modules. Centralized control tool sends commands to robots via Robots communication module. These commands are evaluated by Matlab engine and collected by Matlab communication module. UI module is responsible for graphical user interface which displays situation in the field and allows operator to start some algorithms in test mode or control robots manually.
2. *Matlab algorithm library* [8] provides to analyze the situation in the field and to assign the current roles to the robots based on this analysis. This library is also used by the Matlab Engine to calculate control signals to every robot with regard to its currently assigned role.

An overview of modules and their interaction is illustrated in the Fig. 3.

3.1 Centralized control tool

SSL receiver module In *SSL receiver module* we use standard classes that are supplied with SSL-Vision and are intended to receive and parse packets of Google Protobuf protocol [9]. The result of parsing is transmitted to *Matlab communication module* and *UI module* via Qt signals and classes. If packet from SSL server contains geometry we also generate Qt signal for updating field parameters which are saved as a Qt class and shared between Matlab algorithm library and UI.

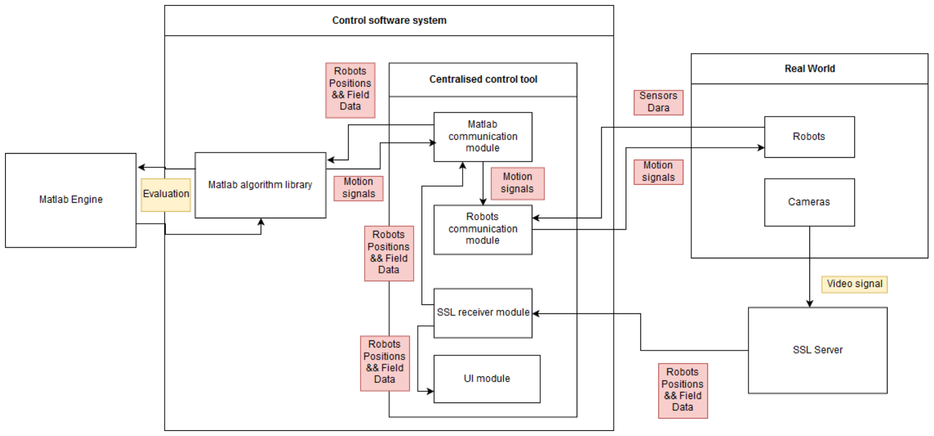


Fig. 4: Software system diagram

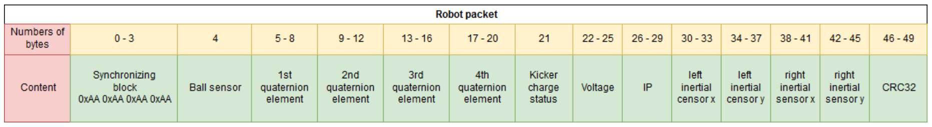
Robots communication module Task of this module is to maintain connection of the centralized control tool with the robots, to receive data from their sensors, and to send control signals to the robots.

All commands to robot and data from him is counted in the robot system of coordinates. Each robot is continuously sending packets with data from its sensors via UDP datagrams. These packets have the special format (Fig. 5), where:

- Ball sensor - byte which indicates if ball is near robot dribbler or not (using robot ball sensor)
- elements of quaternion - indicates orientation of the robot
- Kicker charge status - byte which indicates if kicker is charged and ready for kicking
- Voltage - indicates current voltage from the battery
- IP - IP address of the robot in the network to which it is connected
- left inertial censor x - x axis data from inertial censor which is situated on the left side of the robot
- left inertial censor y - y axis data from inertial censor which is situated on the left side of the robot
- right inertial censor x - x axis data from inertial censor which is situated on the right side of the robot
- right inertial censor y - y axis data from inertial censor which is situated on the right side of the robot
- CRC32 - control sum which was calculated using algorithm CRC32

They are received using QUdpSocket (QtNetwork module[2]) and parsed. Then extracted data is transmitted to *Matlab communication module*.

[2] http://doc.qt.io/qt-5/qtnetwork-index.html



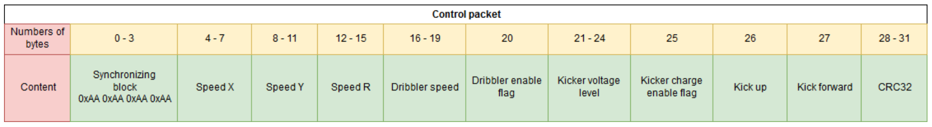
Robot packet													
Numbers of bytes	0 - 3	4	5 - 8	9 - 12	13 - 16	17 - 20	21	22 - 25	26 - 29	30 - 33	34 - 37	38 - 41	42 - 45	46 - 49
Content	Synchronizing block 0xAA 0xAA 0xAA 0xAA	Ball sensor	1st quaternion element	2nd quaternion element	3rd quaternion element	4th quaternion element	Kicker charge status	Voltage	IP	left inertial censor x	left inertial censor y	right inertial sensor x	right inertial sensor y	CRC32

Fig. 5: Structure of packet received from the robot

When *Robots communication module* receives control signals from *Matlab communication module*, it converts them to the special UDP datagram packets (called "Control packets" (Fig. 6)), where:

– Speed X – value from -100 to 100, which indicates percentage of power for motors to move robot along the x axis of its coordinate system.
– Speed Y – value from -100 to 100, which indicates percentage of power for motors to move robot along the y axis of its coordinate system.
– Speed R – value from -100 to 100, which indicates percentage of power for motors to rotate robot (positive value means clockwise rotation of the robot).
– Dribbler speed – value from 0 to 100 which indicates power of dribbler motor in percentages.
– Dribbler enable flag – byte which indicates that dribbler should be turned on/off.
– Kicker voltage level – value from 0 to 30 which indicates power of next kicking action (kicker needs time for charging to be ready for kicking).
– Kicker charge enable flag – byte which indicates if kicker charging should be started or not.
– Kick up – byte which indicates if chip kicker should be activated.
– Kick forward – byte which indicates if straight kicker should be activated.
– CRC32 – control sum which was calculated using algorithm CRC32.

and then transmits them to robots.

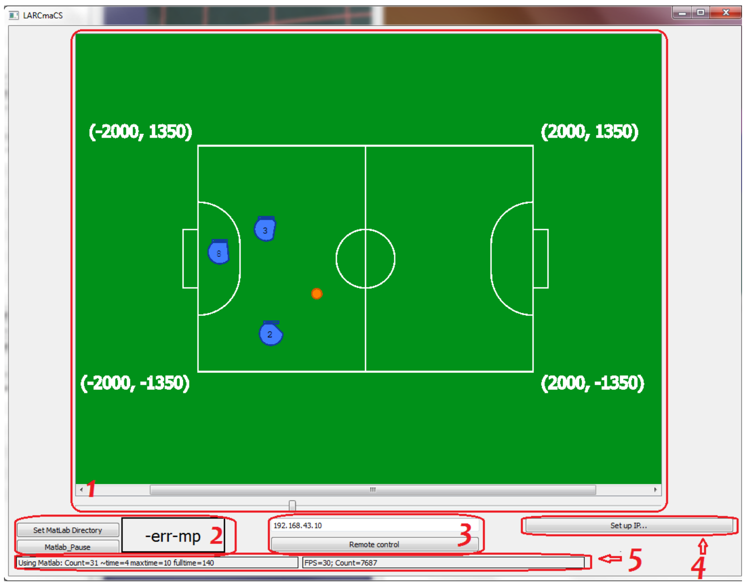


Control packet											
Numbers of bytes	0 - 3	4 - 7	8 - 11	12 - 15	16 - 19	20	21 - 24	25	26	27	28 - 31
Content	Synchronizing block 0xAA 0xAA 0xAA 0xAA	Speed X	Speed Y	Speed R	Dribbler speed	Dribbler enable flag	Kicker voltage level	Kicker charge enable flag	Kick up	Kick forward	CRC32

Fig. 6: Structure of control packet

Matlab communication module This module is responsible for launching Matlab engine, transmitting coordinates of objects on the field to Matlab Engine and extracting control signals for robots from engine after evaluation. We use Matlab C++ Engine API library [10] for getting access to Matlab engine from C++ source code. To underpin calculation of control signals we transmit the coordinates of the ball and robots, as well as sensors parameters to Matlab engine, and then evaluate file ”main.ml”. During evaluation special structure ”Rule”, which keeps control signals for robots, is initialized. After evaluation this structure is exported from Matlab engine and sent to *Robot communication module*.

1. To work with Matlab engine we have introduced class MlData which keeps ”Engine” and data of type ”mxArray” for importing and exporting from Matlab engine (i.e., Yellows, Blues, Balls, etc.). To launch Matlab Engine we use function *engOpen()*. After that it is needed to specify output buffer for Matlab Engine by using *engOutputBuffer()*, set Rule to zero, and to specify the directory where files with our algorithms are accomodated.
2. The coordinates of the balls and robots from the two teams are organized in arrays of doubles. When needed to be transferred the the Matlab Engine, these arrays are first copied to mxArrays. Then they are loaded to Matlab environment by using function *engPutVariable()*. Finally, Matlab engine evaluates ”main.ml” with new loaded data.
3. The control signals calculated in Matlab algorithm library are inserted into the structure Rule Rule. For extracting it from Matlab algorithm library we use function *engGetVariable()*.



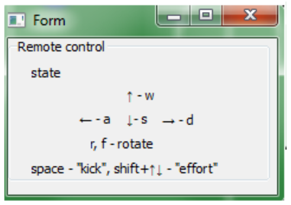
Using Matlab: Count=31 ~time=4 maxtime=10 fulltime=140

FPS=30; Count=7687

Fig. 7: Main window of our application. 1. Game Field 2. Matlab block 3. Remote control block 4. IP Settings 5. Information bars

User interface The main task of UI (Fig. 7) is to show positions of the robots and the ball on the game field, give human operator access to special settings and manual control of robots. The field (1) with scroll sliders and zoom controller is situated in the middle of the user interface. It shows ball, robots of both teams, and has marks with field coordinates in the corners. Matlab settings block (2) is in the left corner of the bottom panel. It allows user to pause or to continue evaluating of control signals, as well as to choose directory where algorithms are located. Remote Control block (3) is at the center of the bottom panel (Fig. 8). It allows to control robots manually using keyboard. Settings for robots IP is situated in the left corner (4) under "setting robots IP" button (Fig. 8). Moreover, some information about connection to SSL server is provided by special information bars (5) at the bottom of the window.

(a) Remote control window



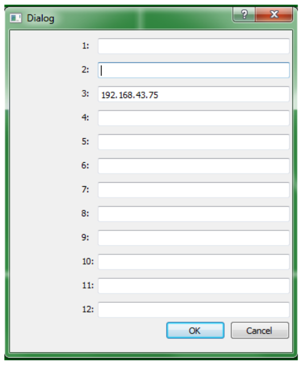
↑ - w

← - a ↓ - s → - d

r, f - rotate

space - "kick", shift+↑↓ - "effort"

(b) IP Settings



1:	
2:	
3:	192.168.43.75
4:	
5:	
6:	
7:	
8:	
9:	
10:	
11:	
12:	

Fig. 8: Extra program windows

3.2 Matlab algorithm library

As it was mentioned in section Matlab Communication module, general scheme of robots control consists of the next steps:

1. receiving new SSL packet with data about robots (Yellows, Blues) and ball (Balls) positions on the field and loading them to the Matlab engine;
2. main.ml evaluating, during which a special structure "Rule" with control signals is being filled;
3. pulling this structure out and sending control signals to the robots.

There are 5 variables which are shared between Centralized control tool and Matlab algorithm library – Blues, Yellows, Balls, Rule, ballInside. The first three variables describe data from SSL, the fourth one contains control signals for robots, the last one defines is ball inside any robot or not. All this variables are declared as double array (except ballInside, which is double scalar), both C++ and Matlab.

At the beginning of "main.ml" evaluation all needed variables and structures are initialized by using *mainHeader* function. During this function global structure *RP* are declared by using loaded data from SSL. This structure will be shared between all algorithms in the future evaluation. Structure *RP* contains the next main fields.

1. Blue – array of structures with information about blue robots (robot presence on the field, robot position, robot angle).
2. Yellow – array of structures with information about yellow robots (robot presence on the field, robot position, robot angle).
3. Ball – structure which contains information about ball position.
4. Pause – flag which controls stopping and starting of evaluation.
5. Rule – array of structures with control signals for robots. (Fig. 9)

During evaluation "RP.Rule" should be filled with calculated control signals. *Rule* has the next fields:

1. "Robot in use" flag – controls do we need to send control signal to this robot or not;
2. Number of robot – number of robot according to SSL Pattern [11];
3. Speed X – robot speed along X-axe of its local coordinate system;
4. Speed Y – robot speed along Y-axe of its local coordinate system;
5. Kick forward flag – controls should robot kick forward;
6. Speed R – robot angular speed;
7. Kick up flag – controls should robot kick up.

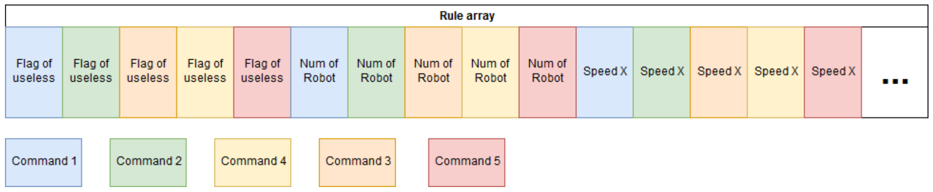


Fig. 9: Rule format description

Control signals are calculated using algorithms presented in section Matlab algorithm library. To stop evaluation we use a special function PAUSE, which switches "RP.Pause" flag. This flag is checked on each iteration at the beginning of main.ml and in case if it is true evaluation is stopped.

3.3 Build configurations

Compilers Initially, the project was developed for Windows platform only for the sole reason that then it would be easily accessible by school students. This motivated us to choose MSVC-compiler [12] for our application. Our current objective is to remaster the software for running on Linux. As a first step to this end, we have already converted the core programs to the to MinGW-compiler [13]. Our software system can be compiled by both of these compilers for Windows platform at this moment.

Architectures Our application needs Matlab. So as not to impose a restriction on the bit-version of the Matlab, both Matlab x64 and Matlab x86 were supported.

Continuous Integration *Travis CI* [14] with *static code analyzer Vera++* [15], which automatically checks codestyle of pull requests, is used for automatic tests. Although Matlab is needed for running our software system in full, but for testing build process of the project and running it, the testing build process calls for only *Matlab Runtime Compiler* [16], which is in free access. Our plans include transition from Matlab to MRC in order to make our software independent of any commercial products.

SSL At the moment we have to support our Centralized control tool with two versions of SSL-vision: old (2012 year) and new (2018 year). The old version is available on both Windows and Linux, while the new version is only available on Linux. We actively use the old one, because of more convenient way to deploy our setup. At the moment we actively try to port new SSL-vision to Windows.

4 Algorithms

The developed algorithms can be categorized into three groups: basic algorithms, advanced algorithms and roles (behaviour patterns). Now we illustrate every group by describing its most important algorithms.

4.1 Main terms

- *SSL coordinate system* – global coordinate system associated with data received from SSL-vision.
- *Robot coordinate system* – local coordinate system associated with robot control model.
- *Robot position* consists of the Cartesian coordinates of robot center and the polar angle of robot in the SSL coordinate system – (x, y, α).
- *Robot velocity* is a vector of velocity in robot coordinate system – $\overrightarrow{v}$.
- *Robot angular speed* is angular robot speed – ω

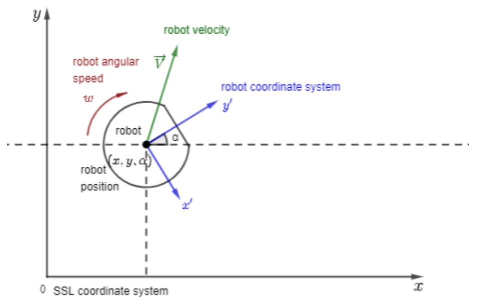


Fig. 10: Main terms

- *Minimal robot speed* is a minimal speed at which robot starts to move – v_{min}.
- *Minimal angular robot speed* is a minimal speed at which robot starts to rotate – ω_{min}
- *P, I, D* – are the proportional, integral, and differential coefficients of the considered PID-controller.

4.2 Basic algorithms

MoveToPoint This algorithm controls robot's moving to the destination point. (Fig. 11)

Input: robot position, destination point.

Output: robot velocity.

P-controller is used to calculate the magnitude of robot velocity:

$$V = V_{min} + P * |\overrightarrow{S}|$$

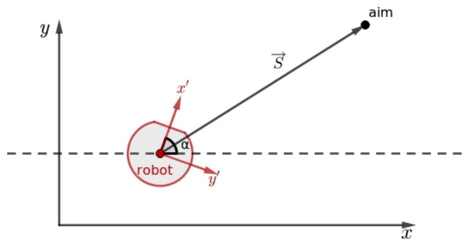


Fig. 11: Algorithm MoveToPoint

To translate direction vector $\overrightarrow{s}$ to robot coordinate system we use the next formulas:

$$\overrightarrow{s} = \frac{\overrightarrow{S}}{|\overrightarrow{S}|}$$

$$\overrightarrow{u} = (cos\alpha, sin\alpha)$$

$$V_x = V * (s_2 * u_1 - s_1 * u_2)$$

$$V_y = V * (s_1 * u_1 + s_2 * u_2)$$

RotateToPoint This function controls robot rotation to the destination point. (Fig. 12)

Input: robot position, destination point.

Output: robot angular speed.

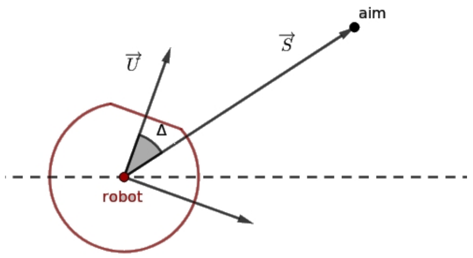


Fig. 12: Algorithm RotateToPoint

To rotate to the destination point we use P-controller, which looks like:

$$w = sign(\Delta) * w_{min} + P * \Delta$$

where Δ – difference between current robot angle and destination point angle in SSL coordinate system.

GoAroundPoint This algorithm drives the robot around a given point at a given distance and controls that robot is rotated to the point (looking after the point). (Fig. 13) If initially the robot is not at the requested distance from the point, the algorithm preliminarily drives the robot to this distance.

Input: robot position, center of rotation, radius of circle.

Output: robot velocity, angular robot speed.

To rotate around point we use algorithm RotateToPoint with some modifications. This algorithms is used in cases of circles with small radius, therefore the high accuracy is necessary, so we use PD-controller. Formula of angular speed is:

$$w = sign(\Delta) * w_{min} + \Delta * P + (\delta - \Delta) * D$$

where Δ − is difference between desired and current angle at current step, δ − difference between angles at previous step.

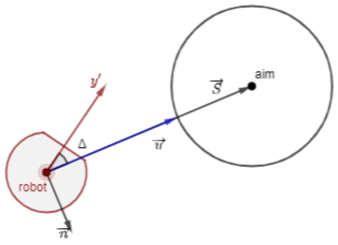


Fig. 13: Algorithm GoAroundPoint

The next point where robot should move is calculated as:

$$\overrightarrow{S} = \frac{\overrightarrow{S}}{|\overrightarrow{S}|}$$

$$point_x = x + c_1 * n_1 + c_2 * L * s_1$$

$$point_y = y + c_1 * n_2 + c_2 * L * s_2$$

where $\overrightarrow{n}$ - normal vector, c_1 and c_2 − coefficients, $L = |\overrightarrow{S}| - R$.

To calculate robot velocity we use MoveToPoint algorithm to $(point_x, point_y)$.

Starting from this moment we combine robot velocity and robot angular speed in function *MoveToWithRotation*, which moves and rotates robot to the destination point simultaneously. In the further description we will refer to it.

4.3 Advanced algorithms

TakeAim This function drives robot to the line, which connects ball and aim. This function controls, that robot stops at the desired distance from the ball on this line. (Fig. 14)

Input: robot position, aim coordinates, and parameters for function GoAround-Point.

Output: robot velocity, angular robot speed.

If robot is in desired point, it rotates to the point using RotateToPoint. In other case, GoAroundPoint is called.

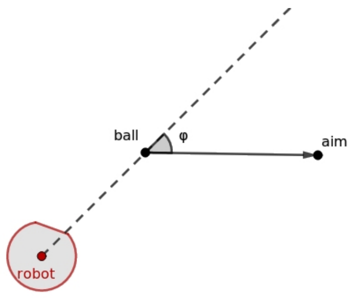


Fig. 14: Algorithm TakeAim

Catch ball This function allows robot to take a pass from other robot. The main idea is as follows. As soon as the robot is hit by the incoming ball, do not catch the rest the robot should move with none-zero velocity, which is co-directional with ball velocity and depends on it. In this case kinematic energy will decrease, and ball will not bounce off far. (Fig. 15)

Input: robot position, ball position.

Output: robot velocity, angular robot speed.

In case if ball speed, calculated from ball positions in previous and current frames, greater than minimal value (constant), estimated ball trajectory is calculated. In other case robot just rotates to the ball using RotateToPoint function.

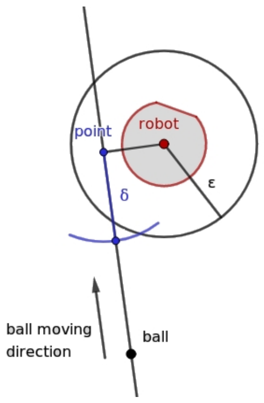


Fig. 15: Algorithm CatchBall

Point, which is a foot of perpendicular from the robot center to the ball trajectory, is calculated. It is considered, that pass was taken to other robot in case if distance from the robot center to this point is less than ϵ. If distance to this point is greater than δ then robot moves to this point using MoveToPoint. In other case a new point is calculated, which got from the current point using movement along the ball trajectory, proportionally to the ball velocity. Robot moves to the new point and turns to the ball.

BuildPath This function builds path between starting point and destination point. There are several obstacles (defined as circles) on the plane between those points. (Fig. 16)

Input: starting point S, destination point F, array of obstacles (obstacle is a pair of center of circle and radius), parameter *step*.

Output: array of points of path.

In algorithm segment SF is considered. If it doesn't cross any obstacle, then the path is SF. In other case, from all obstacles which were crossed, algorithm chooses the nearest one to the point S using Euclidean metric. Then algorithm calculates point C on the line, which is perpendicular to the SF and crosses the center of the nearest obstacle. C is located at distance *step* from the obstacle (segment OS). In case *step* is positive then $\overrightarrow{OC}$ is turned clockwise from $\overrightarrow{SF}$. In other case – counterclockwise from $\overrightarrow{SF}$. Then algorithm calculates path recursively for SC and CF segment.

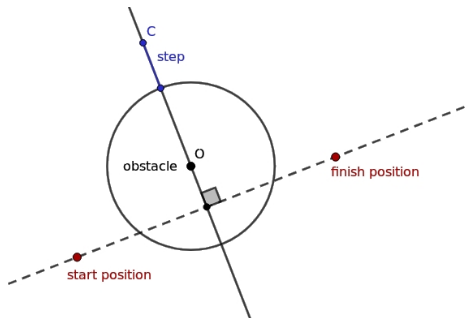


Fig. 16: Algorithm BuildPath

In practice we limit depth of recursive call. Therefore if path wasn't constructed, we assume that point is unattainable.

MoveToAvoidance This function controls movement of the robot to the destination point with obstacle avoidance. Obstacles are defined as circles.

Input: robot position, destination point, array of obstacles (obstacle is a pair of center of circle and radius).

In algorithm robot is a material point, therefore to avoid obstacles we increase obstacles radius by robot radius. It may happen, that current robot position A is inside obstacle (for example, due to equipment error). To avoid incorrect path planning, we use a point A' which lies at some distance from the obstacle on the line OA, where O is obstacle center and A lies between O and A'.

On each step we calculate two paths – with positive step and with negative step using BuildPath function, choose the shortest path and move to the next point using MoveToPoint.

4.4 Behaviour models

Goalkeeper This function describes goalkeeper behaviour. Goalkeeper is moving along the goal and its trajectory is a line. If estimated trajectory of the ball is crossing the goal, robot is moving to the point of their intersection. Estimated trajectory is a line calculated from previous ball positions. (Fig. 17)

Input: robot position, ball position, goal center, goal vector $\overrightarrow{N}$.

Output: robot velocity.

To calculate intersection point of goals and ball trajectory we use the next formula:

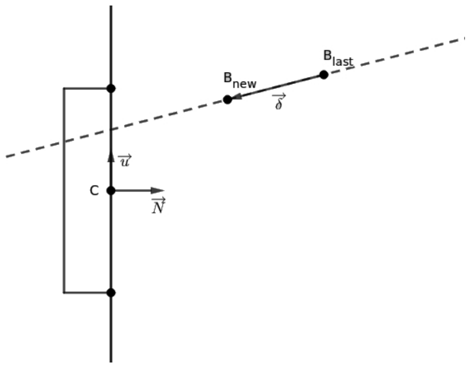


Fig. 17: Algorithm Goalkeeper

$$t = \frac{\overrightarrow{CB_{new}} \wedge \overrightarrow{u}}{\overrightarrow{\delta} \wedge \overrightarrow{u}}$$

$$point_x = B_{new_x} + t * \delta_1$$

$$point_y = B_{new_y} + t * \delta_2$$

To move to the point of intersection we use function MoveToPoint with PD-controller. In case of axe which is perpendicular to the goal, the speed is the most important thing, in case of parallel axe stabilization of robot is necessary. Therefore robot velocity along axes X and Y in global coordinates is calculated in different ways with different P and D coefficients. After this step velocities are transformed to robot coordinate system and summarized.

Attacker This function controls attacker behaviour. It provides kick in aim direction.

Input: robot position, ball position, aim position.

In this algorithm every combination of robot position, ball position and aim position belongs to one state of four possible states. Depending on what state describes the current arrangement of these objects robot makes a decision on further actions.

– *State 1:* robot is far from the ball. It means that distance from the point to the ball is greater than defined constant. In this case robot moves to the ball using function MoveToWithRotation, until distance to the ball will reach defined constant.
– *State 2:* robot is not in state 1, but robot doesn't take aim. It means that ball is not on the line between robot and aim. In this case we use algorithm GoAroundPoint, which provides robot movement around the ball until robot will reach desired point.
– *State 3:* robot is not in state 2, ball is on the line between robot and aim, and robot is near to the ball. In this case robot moves to the ball until touch it. In state 1 robot doesn't move closely to the ball, so as not to touch it when aiming.
– *State 4:* ball is inside robot and robot has taken the aim. To check is ball inside robot we use data from robot ball-sensor. In this case robot kicks the ball.

5 Acknowledgements

We wish to express our deep gratitude to prof. Alexey Matveev (SPbU) for his review and useful advice for the scientific sections of this paper. We also would like to thank Iakov Kirilenko (SPbU) for his enthusiastic encouragement and professional support. Special thanks should be given to previous research group (circa 2012-2016), which developed Robocup-SSL game for two-wheeled robots based on Lego, Arduino, and TRIK – Ilya Shirokolobov, Ruslan Sevostyanov, Kirill Ovchinnikov – for their responsiveness and advertency.

Our grateful thanks are also extended to JetBrains Research, which encourages leading students from our team with a scholarship. Finally, we wish to thank Presidental Lyceum of physics and mathematics 239 and CybetTech Labs Co Ltd for their valuable technical support on this project.

References

1. Cybernetic constructor TRIK, official webpage, `https://trikset.com` official English webpage, `http://blog.trikset.com/p/eng.html`
2. Russian inter-university robotics school, official webpage, `https://vk.com/roboschool_vlg`
3. Robofinist, official webpage, `https://robofinist.org`

4. Qt, official webpage, https://www.qt.io
5. MATLAB, official webpage, https://www.mathworks.com/products/matlab.html
6. Centralized control tool repository, https://github.com/robocup-ssl-russia/LARCmaCS
7. SSL Vision system, official repository, https://github.com/RoboCup-SSL/ssl-vision
8. Matlab algorithm library repository, https://github.com/robocup-ssl-russia/MLscripts
9. Google Protocol Buffers, official webpage, https://developers.google.com/protocol-buffers
10. Matlab C++ Engine API, official webpage, https://www.mathworks.com/help/matlab/matlab_external/engine-c-api-1.html
11. Official SSL Vision standard pattern, http://wiki.robocup.org/images/9/96/Small_Size_League_-_Standard_Pattern_2011.pdf
12. Microsoft Visual C++ compiler, official webpage, https://visualstudio.microsoft.com/ru/vs/features/cplusplus
13. MinGW compiler, official webpage, http://www.mingw.org
14. Travis CI, official webpage, https://travis-ci.com
15. Static code analyzer Vera++, official webpage, https://bitbucket.org/verateam/vera/wiki/Home
16. Matlab runtime compiler, official web page, https://www.mathworks.com/products/compiler/matlab-runtime.html