PARSIAN 2018
Extended Team Description Paper

Mohammad Mahdi Rahimi, Mohammad Mahdi Shirazi,
Mohammad Amin Najaf Gholyan, Fateme Hashemi Chaleshtori, Nadia Moradi,
Kian Behzad, Seyed Hamidreza Roodabeh, Ali Gavahi, Fateme Farokhi
Moghadam, Seyed Ali Ghazi Asgar, Yasamin Alizadeh Gharib, Mahshid
Memarian, Amir Hadi Tavakoli, and Mohammad Azam Khosravi

Electrical Engineering Department
Amirkabir Univ. Of Technology (Tehran Polytechnic)
424 Hafez Ave. Tehran, Iran
{mmrahimi,mhmmdshirazi,hashemi96,nadiamoradi,kian.behzad,hr.roodabeh,
aligavahi,fateme.fmoghadam,alizadeh_yasi,m.a.khosravi}@aut.ac.ir
http://www.parsianrobotics.aut.ac.ir

Abstract. This paper presents Parsian's hardware elaboration, the software architecture and all improvements that have been made since last year, including useful innovations in hardware, e.g. new ball detection sensor, debugger module and robot's fault recovery. Noteworthy enhancements in software such as micro-service architecture by ROS, open loop motion correction, motion profiler and new obstacle avoidance strategy are described.

Keywords: microservice, ROS, motion control, fault recovery

1 Introduction

Parsian Robocup Soccer team formed in 2005 by Electrical Engineering Department of Amirkabir University of Technology, and has been working on small size league since then. This team aims to design and build SSL robots, compatible with international RoboCup competition rules as an engineering project. This team has been qualified for twelve consequent years for RoboCup SSL, and participated in 2008 to 2017 RoboCup competitions. Parsian's most significant success is first place in RoboCup 2012 and RoboCup 2013 technical challenges and also fourth place in RoboCup 2012 and 2017.

1.1 Team members

Mohammad Azam Khosravi: Control Theory, Supervisor
Mohammad Mahdi Shirazi: Control, Firmware, Team Leader
Mohammad Mahdi Rahimi: AI Software
Mohammad Amin Najaf Gholian: Electronic, Mechanic
Fateme Hashemi Chaleshtori: AI Software
Nadia Moradi: AI Software
Kian Behzad: AI Software
Seyed Hamidreza Roodabeh: AI Software
Ali Gavahi: AI Software
Fateme Farokhi Moghadam: Electronic
Seyed Ali Ghazi Asgar: Electronic
Yasamin Alizadeh Gharib: Electronic
Mahshid Memarian: Mechanic
Amir Hadi Tavakoli: AI Software

2 Hardware

2.1 Mechanic

Motor and Gearbox. This year the gear ratio has been changed from 3.6:1 to 2.5:1, to increase the maximum velocity. The major innovation is changing the angle of the wheels' cone, so that the robot's center of mass can be lowered and solenoids can be flattened as well. In this concept, the chip's rocker shaft has been raised since it has a higher transient impulse power, and chip solenoid core collides to the center of percussion of chip shaft.

Dribbler and Spin. At the back of the dribbler, a spring-damper system has been placed. According to the mass of the dribbler and the ball, it will be over-damped at high speeds (over 10m/s). Since the relative stiffness of the dribbler causes spin disruption and resonance in the movement of the ball, this is not the perfect solution. So the chosen damper, has the ability to be calibrated for different carpets.

2.2 Electronic

Kick Sensor. To detect the ball, infrared transmitter and infrared receiver sensors are embedded in both sides of the robot's dribbler. In the previous design, a typical IR photo-diode sensor has been used that was highly affected by ambient light. In the new design, a typical IR LED is used, along a TSOP1238 to receive the infrared signals.
Each member of TSOP12xx series is sensitive to different frequencies of IR spectrum; the xx shows the frequency that the sensor detects. For generating the IR signals at 38Khz, a 555 Timer IC is used in astable multi-vibrator mode. In this mode, it works as a free running oscillator and generates approximately 38 kHz square waves.

Debug Mode. Last year essential data such as battery voltage, wireless data-lost ratio, motors fault and other failures has been sent to communication module. This year, besides that, debug part has been added to send robot's parameters including wheels speed, control variables, etc. for monitoring and analyzing.

Fault Recovery. Theres two major methods in control for fault recovery procedure; (1) active fault recovery that means the control system is redesigned when a fault happens and (2) passive fault detection that means the control system is robust enough to tolerate probable fault.
This year an active fault recovery has been implemented for motor-based faults. First of all if a motor connection has been unplugged or the motor has some failures like hall-sensor faults, the systems kinematic is switched to an appropriate three-wheel kinematic; so the robots able to move, either to play the game with less efficiency or just to reach the legal substitution place in the field.
The second is the encoder fault that can happen because of different reasons like unplugged encoder socket or electrical failure. In this case the velocity measurement reference, is changed from encoder to the motors hall sensors.

Optical Flow. PAN3401 is a CMOS process optical mouse sensor single chip with a PS/2 interface that serves as a non-mechanical motion estimation engine for implementing a computer mouse. A CMOS image sensor sends each image to a digital signal processor (DSP) for analysis. The DSP operating speed is 18 MIPS. It can detect patterns in the images and compare them to the previous image. Based on the change in patterns over a sequence of images, the DSP determines how far the sensor has moved during a certain time. The movement measurement frequency is 100Hz. This sensor will be used in future works as a local reference of linear velocities.

3 Control

Open-Loop Motion Correction. In order to achieve perfect motion control, a new learning-based method has been implemented which fixes open-loop motion errors. In polar coordinates, the robots' movement is expressed as direction angle (θ), velocity (V) and angular velocity (ω). In this method, two offsets have been added to θ and Ω; linear velocity is accurate enough and remains without offset. At first, a PSO method was implemented with a complicated cost function to optimize multiple variables simultaneously; but after reviewing the results, it was realized that these two parameters (θ and ω) are almost independent and a simple method can be employed separately, for each parameter.
One efficient method that can be utilized to optimize a parameter, is *Error Back Propagation*. It's mainly used in finding weights of neural networks. The result of this method was satisfying; one instant is depicted in Fig. 1

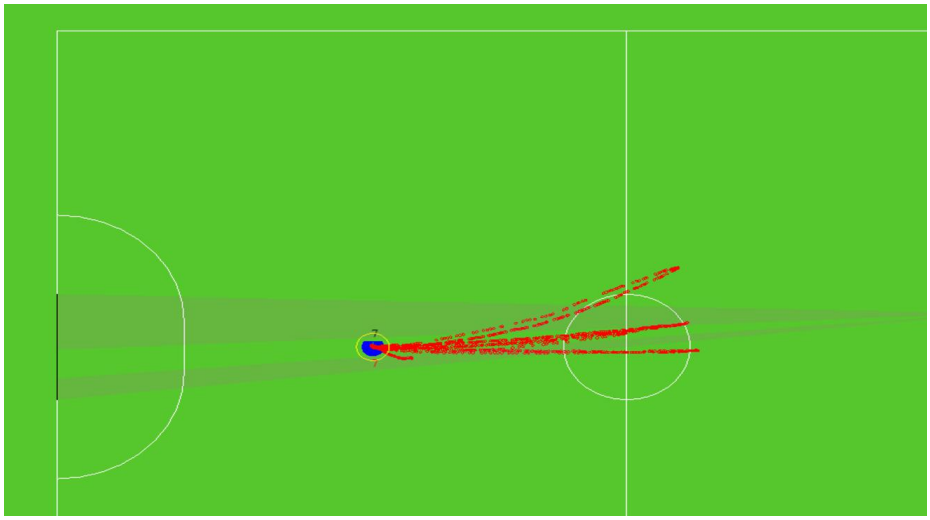

Fig. 1. Open-loop motion learning

Angle-base Deceleration. The robots maximum deceleration depends on the movement angle. Previously a constant deceleration had been used for all angles, but this year it was extended to three different decelerations, one for each of the forward, backward and normal angles. Desired deceleration for each angle will be calculated by a weighted average, depending on movement angle (Fig. 2 and 3).

Motion Profiler. Motion profiler is a module which moves a robot in different distances, angles, and directions. Then it records vision data, robots motion data and dispatched commands, also extracts useful information from these raw data (Fig. 4 and 5).

1. Robot velocity-time table of each motion
2. Remaining distance-time table of each motion
3. Command velocity-time table of each motion

Then it calculates some useful information including:

1. Total delay by calculating time shift between speed and command
2. Time needed for a robot to move from one point to another, with four-dimensional regression on the raw data

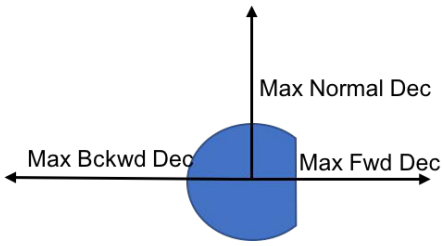

Fig. 2. Deceleration in different angles

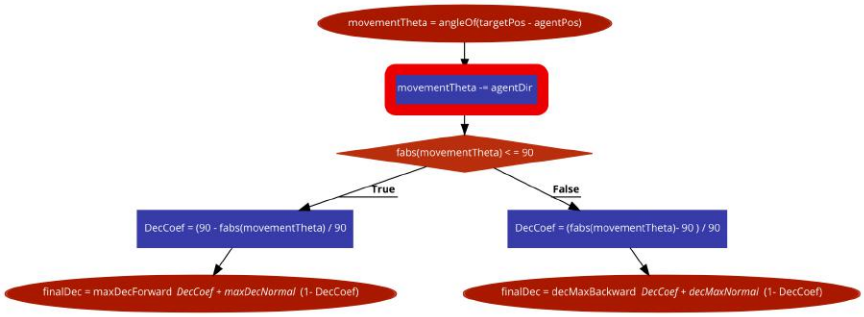

Fig. 3. Flow-chart of calculating desired deceleration

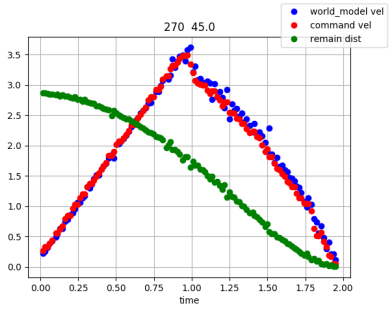

Fig. 4. Raw data which is collected for short path

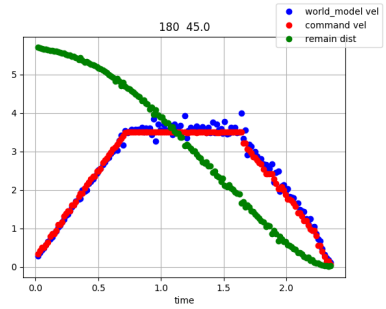

Fig. 5. Raw data which is collected for long path

Obstacle Avoidance. Parsian has been using *ERRT* algorithm for years that is mostly used in real-time approaches. In order to decrease the number of collisions and also minimize the travel time, a new approach for obstacle avoidance has been developed. This method does not intend to change *ERRT* implementation or propose a new way to avoid obstacles; its rather about how to define the obstacles space. This method is based on opponent robots capabilities:

1. Maximum Deceleration
2. Maximum Velocity
3. Agility Factor [1]
4. Current Velocity

A probability area can be chosen for a certain upcoming time interval. If the probability area for opponent robots and future position of our robot overlaps, then the obstacle avoidance try to avoid that common space. (Fig. 6)

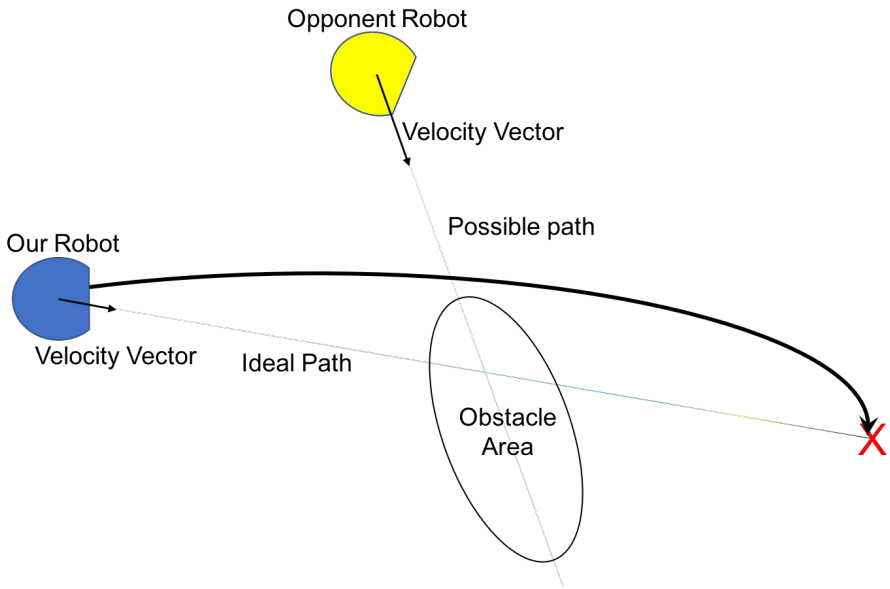

Fig. 6. Generated path, considering overlap area

[1] Agility Factor represents by how many degrees, the movement direction of a robot can change, during a specified time in its maximum velocity.

4 Software

4.1 Microservice

Parsian Software has been splitted into different components and the monolithic based architecture has been altered to distributed. The new architecture has a lot of benefits such as:

- **Agility**: New features and products can be added more quickly.
- **Independent Deployment**: The components are easy to understand and modify; this can help a new team member become productive quickly.
- **Polyglotism**: Team has freedom to choose technology and programming language that is best suited for a particular functionality.
- **Better Fault Isolation**: If one component fails, there's a less chance to impact other ones, and they will continue to work.
- **Testability**: Test surface is smaller than that of monolithic applications thus its easier to be tested.
- **Easier Analysis**: The transferred messages leads to analyze the data in a more convenient way.

4.2 ROS

Robot Operating System (*ROS*) is a collection of software frameworks for robot software development. ROS provides services designed for heterogeneous computer clusters such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture, where processing takes place in nodes. Despite the importance of reactivity and low latency in robot control, ROS itself, is not a real-time OS (*RTOS*), though it is possible to integrate ROS with real-time code. There are three types of software In ROS ecosystem:

1. Language and platform-independent tools
2. ROS client library such as roscpp, rospy, rosjava, etc.
3. Packages containing application-related code by using ROS client libraries

In Parsian project the control and AI nodes are written in C++ using roscpp. GUI, test nodes and tools like profilers are implemented with python scripts (rospy).

4.3 Architecture

Package. Parsian Stack (ROS repository), packages and their dependencies graph, are shown in Fig. 7.

Packages are the first-level directory that separate source codes; so to implement a package for the first time, debug, upgrade or even re-factor it, only that particular package and its dependencies need to be changed.

First-level dependencies of Parsian packages to ROS packages are shown in Fig. 8:

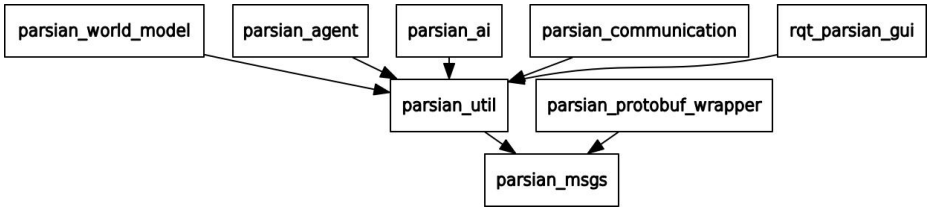

Fig. 7. Parsian packages graph

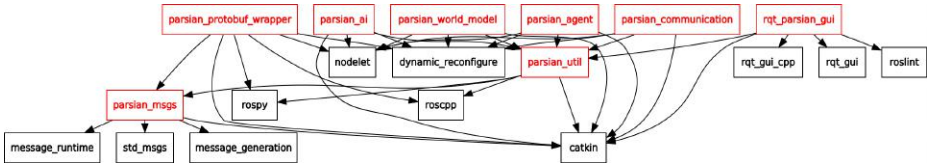

Fig. 8. Parsian packages and first-level dependencies graph

Node. Each package has a number of nodes inside, which are actually executable files. Graph of nodes that run a game with one agent and AI is illustrated in Fig. 9. Nodes can be implemented with different languages and environments. They are executed separately, so when one node freezes or crashes, other ones still execute without any problem. 8:

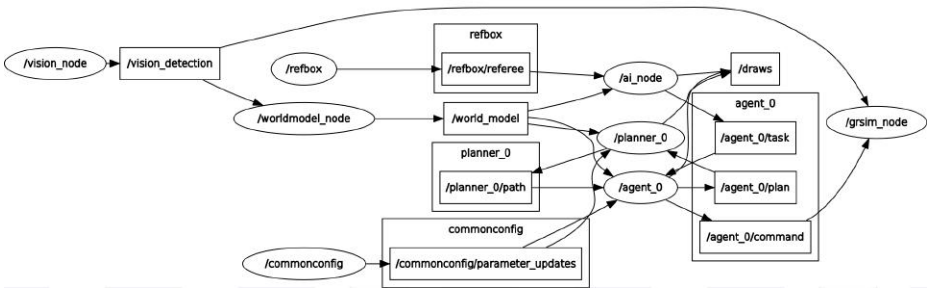

Fig. 9. Nodes and Topics graph for one agent with AI

Message. The most important part of designing a distributed service, is defining the messages that are going to be passed between nodes. This project tries to use pre-defined ROS messages as much as possible, to make it easier for integration with currently implemented services and nodes. You can find Parsian messages, in its repository: `https://github.com/ParsianRoboticLab/parsian_msgs`

4.4 Implementation

Latency and Delay. ROS framework has lacked real-time node and services, mostly because of the latencies that serializing and parsing of messages cause, although it is not always necessary to have low latency, in GUIs for instance. The solution to resolve this latency is using *nodelet* for c++ nodes. (Table 1)

Table 1. Latency of Message Passing

Measured parameter	Nodelet			Node		
	Mean	Min	Max	Mean	Min	Max
world model (heavy message)	0.039 ms	0.073 ms	0.51 ms	0.134 ms	0.885 ms	3.398 ms
robot command (light message)	0.024 ms	0.058 ms	0.42 ms	0.064 ms	0.565 ms	3.232 ms
All message (critical path*)	0.494 ms	0.163 ms	3.220 ms	3.4634 ms	0.490 ms	32.763 ms

Nodelet. Nodelets are a type of ROS nodes, designed to run multiple nodes in a single process, with each node running as a plugin with the help of *pluginlib*. At the very first glance, nodelets are exactly like nodes, but there is a fundamental difference between them; nodes are executable and can be run separately, whereas nodelets are not executable; they are just a software component that are loaded as plugins on a special node, called *nodelet manager*.
Regular nodes use TCP protocol (although if nodes run on one single PC, the shared-memory protocol supersedes TCP protocol). This works fine in most cases, but if you have multiple processes that need to use messages that contain large amounts of data exchange, then packaging the message, sending and unpacking it can take too much time. If the two processes are on the same computer, it is quicker to just send a pointer to that data rather than sending the data itself via shared-memory.
Nodelet only works for processes on the same computer, since a pointer for one computer doesn't make sense for another ones. Nodes, on the other hand, can work on connected computers over network, since you're sending the actual data. To fix this problem a node is written for each nodelet as its nodelet manager. So the nodes have become entirely independent and its possible to continue work over networked computers.

Multi-agent. Since execution of robots skill is independent to each other, they can be run in parallel; so there is a separate node for each robot, in which the robot's operations and tasks is handled. *Agent-Node* generates its own robot's commands and send them via ROS message.

Path Planner Node. Path planner node is separated from main agent nodes, and run beside them. Advantage of this separation is:

1. Path planning is run in a different process, parallel with agent nodes, so there would be more *CPU Utilization*.
2. If a task takes too much time to complete, it wont effect main control task, and command for robots will publish at a same frequency.
3. Development and debug of planner can be easily done by sending and receiving messages.

Strategy Server. There are strategy files for static plays in free-kicks[3]. Based on situation, a strategy should be chosen to run. Loading strategies, finding the best one that matches the game situation, and also analyzing the result of the selected strategy execution, are gathered into a node named *strategy-server*. The routine of strategy selection service:

1. In free kicks, AI node sends a request to *strategy-server node* that includes game state and a list of players.
2. *strategy-server node* first collects some strategies that can be executed considering the data received from AI node.
3. Chooses the best one, based on its history, and sends the strategy to AI node as a response.
4. Evaluating the executed strategy is implemented in AI node and the result is sent back to strategy-server to be analyzed. These data is recorded and will influence the next strategy selection.

GUI and rqt *rqt* is a Qt-based framework for GUI development in ROS. rqt makes it easier to manage the various windows on the screen at the same time. A GUI is developed in a separated node for each widget. All GUI processes run separately and use message passing to communicate with other nodes. (Fig. 10)

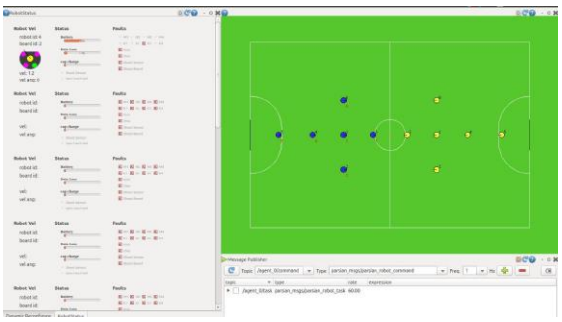

Fig. 10. Robot status and monitor widget in rqt

5 Conclusion

As Robocup SSL rules has changed recently, AI and behavior will be updated to be compatible with new rules.
Also major work on passing and receiving in dynamic plays, started in 2016[2], will be used in a real game, since time calculation with new *Kalman Filter* and *motion profiler* is now accurate enough to execute passing.
Last year, auto-profiler and log analyzer were proven to be very useful. In Iran Open and Robocup 2017[1], auto-profiling and log analyzing, greatly reduced the amount of team setup time, which allowed the team to focus more on strategy planning. Although the main software that was built in 2008, and improved each year, has given successful competition result for the last years (the result summarized in Table 2, changing old monolithic software architecture to distributed in this year, was another enormous work that makes a great advantage to develop, test and debug system. Also the huge number of open-source project helped not to reinvent the wheel.
This year, Parsian hardware's changes aim to stabilize robots with detecting and monitoring their faults. Next steps are online configuration of robot's low-level parameters, and then auto-calibrating the controller's parameters along the game.
In control part, motion controller is improved greatly by improving *Kalman Filter*, *open-loop calibration* and *motion profiler*; most important outcome is accurate timing for robot's movement.

Table 2. Parsian Achievements

Year	Result
RoboCup 2014	Round Robin
RoboCup 2015	Lucky Loser
RoboCup 2016	Lucky Loser
RoboCup 2017	4th Place

References

1. Rahimi, M., Shirazi, M., Arfaee, M., Najaf Gholian, M., Zamani, A., Hosseini, H., Hashemi, F., Moradi, N., Ahsani, A., Jafari, M., Zahedi, A., Abdoullahi, P., Zolanvari, A., Khosravi, M.: Parsian 2017 Extended Team Description Paper for RoboCup. (2017)
2. Rahimi, M., Shirazi, M., Dajkhosh, P., Zolanvari, A., Arfaee, M., Kazemi H., Abbasi, A., Saeidi, A., Khosravi, M.: Parsian 2016 Extended Team Description Paper for RoboCup. (2016)
3. Zolanvari, A., Shirazi, M., Dajkhosh, P., Naderi, M., Arfaee, M., Behbooei, M., Kazemi H., Tazimi, E., Rahimi, M., Saeidi, A.,: Parsian 2015 Extended Team Description Paper for RoboCup. (2015)