NAMeC - Team Description Paper Small Size League RoboCup 2025 Application of Qualification in Division B

A. Calugi, B.Chew, P. Félix, J. Gautier, C. Godinat, C.Labbé, J. Lindois, T.W. Menier, C.A. Vlamynck

IUT - Université de Bordeaux, Gradignan, France
`wanchai.menier@gmail.com` (corresponding author)

Abstract. The following paper presents the advancements of NAMeC, french robotics team which has participated four times in the SSL league. We will explain in detail the software improvements in terms of robot control, such as the new decision-making system and optimal pass computation. Some improvements in the firmware will also be reviewed.

Keywords: RoboCup, Small Size League, embedded systems, path planning, decision making, optimal pass location

1 Introduction

Since joining the RoboCup Small Size League in 2018, the team has participated in four major competitions: Montreal, Sydney, Bordeaux, and Eindhoven. The last competition marks a significant milestone, as we transitioned to a fully student-led organization. As the electronical structure of the robot is being renewed [1], the team has kept the current electronics. The software has been improved with the implementation of a decision-making system, named "Big-Bro", which has been used during the last competition. A score-based best pass location algorithm is also presented, which we plan to use to make use of passes in a match. Additionally, we have introduced a new obstacle avoidance method, that proved effective at last RoboCup. This paper also presents how we use functionalities of the radio module used for communication, the nRF24L01+, to receive feedbacks from the robots while avoiding packet collisions.

2 software

2.1 Decision-making system Big Bro

In a multi-robot system, efficient communication and decision-making are essential to achieving coordinated behavior. To facilitate this, we implemented “Big Bro”, a centralized message-based decision manager that allows robots to exchange information and coordinate their actions dynamically.

Message passing mechanism Big Bro functions as a message pipe, where all robots continuously read and write messages. Each robot listens for messages addressed to it and can publish its own messages, allowing for real-time data exchange.

Decision-making process Depending on the current game state and robot capabilities, Big Bro assign tasks to robots, such as *defending the goal*, *intercepting the ball*, or *passing to a teammate*.

Example: Searching for a Receiver When one of our robot owns the ball, and the opponent goal is being defended by enemies, the robot will send a message to Big Bro, asking for a teammate to pass the ball to. Big Bro will then select the best teammate to pass the ball to. He will then assign the passing strategy to the robot with the ball, and the receiving strategy to the selected teammate.

The attacker target calculation is done by casting robots shadows on the enemy goal line (see figure 1), and retrieves the resulting free spaces where we can score a goal. In the case where the entire goal line is covered by the shadows, the searching for a passer message will be sent to Big Bro.

Future Enhancements While the idea of Big Bro is simple, it has proven to not be very effective in practice because of the way we implemented it. It’s working correclty for simple tasks like strategy attribution depending on game states, or assigning attacker to be the closest bot the ball. But when it comes to tasks involving message exchange between multiple robots, it becomes very difficult to code and maintain considering the number of states that each robot can take. So we plan on rewriting Big Bro to be more efficient and effective in the future.

2.2 Obstacle avoidance: exploration with left and right recursive search

Before the competition of 2024, the team has implemented a new obstacle avoidance algorithm which has proven to be effective at low driving speeds and with

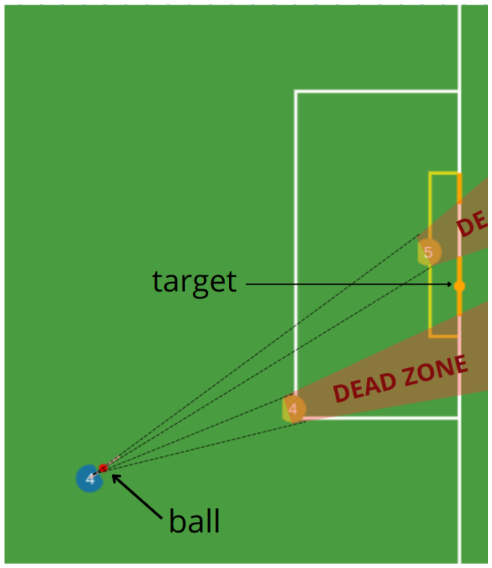


Fig. 1: Attacker target choice example

moving targets. This section will go in detail about how the implementation works.

When the robot wants to move to a specific position, he first checks if any other obstacle isn't in his way (direct line between him and the target), if there isn't an open path, it initiates a pathfinding algorithm to explore alternative routes. This exploration phase involves adjusting the robot's direction by incrementally rotating the vector facing the target, to the left and to the right, until a free angle is identified. The exploration tree is a binary tree with a branch going to the left and the other to the right of the robot.

– **Obstacle Detection and Rotation** Upon detecting an obstacle, the robot begins exploring by rotating incrementally in one direction (e.g., left) (see figure 2 for a visual example). After each small rotation, it evaluates whether the new direction results in a clear path of a short length. The robot checks if any obstacles intersect the trajectory. If no obstacles block the path, the angle is marked as "free", and the robot proceeds along this trajectory. Otherwise if an obstacle remains, the robot continues rotating further in the same direction (left or right) until a free path is found or the maximum angle limit is reached.
– **Recursive Exploration**
When a free path is identified, the algorithm recursively evaluates the next step from the new position. It continues checking whether this position brings the robot closer to the target while considering any new obstacles in its way. The exploration process is repeated until the target is reached or the maximum iteration number is reached.

- **Trajectory Smoothing**
  Once a path is calculated, it is smoothed to eliminate unnecessary waypoints, retaining only significant points to optimize the trajectory. It is performed by iterating over the points constituting the path, from the target to the robot, and check for each points if the robot can reach it without hitting an obstacle. If it can, the point is removed from the path, otherwise it is kept and we continue to the next point considering the current one as a new reference.
- **Optimal Direction Selection**
  If both left and right explorations yield free paths, the algorithm selects the optimal path based on distance to the target. If no fully clear path is found, the robot moves along the trajectory that gets it closest to the target.
- **Continuous Recalculation**
  The algorithm runs continuously, recalculating the best trajectory in real-time. Because of this, we just need to set the robot target to be the first point of the path found by the algorithm.

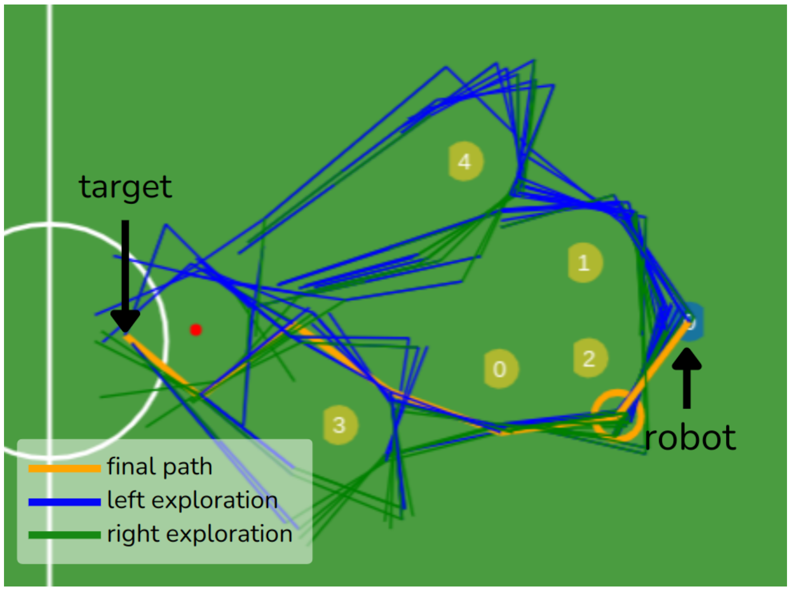
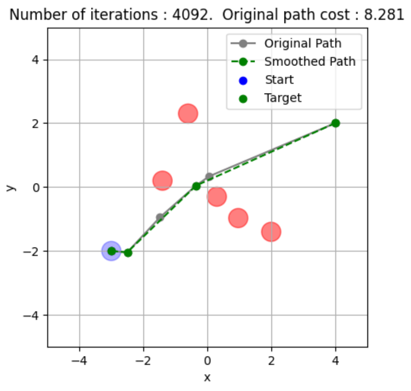


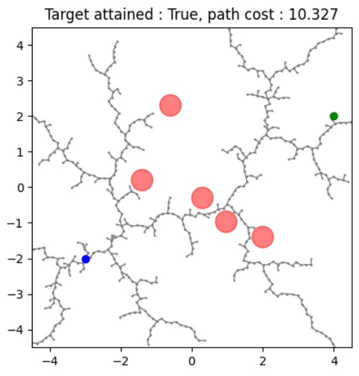
Fig. 2: Path finding example

Comparison with Informed RRT* This obstacle avoidance algorithm has proven to be sufficient in practice. To compare its performance, we have decided to implement a modification of the Informed RRT* path finding algorithm described by team KIKSbot [3]. Since RRT* is a popular choice in the SSL communinity, choosing this algorithm seems to form a good comparison basis.

The Informed RRT* search was parameterized to perform 800 iterations, but because of its stochastic nature, a path might not be found during these steps. In a real-life context, the path finding is run continuously during a match, so the search has been run 20 times in order for the algorithm to give us the best result and lowest-path cost (in terms of distance traveled). Note that no smoothing was performed on the Informed RRT* result.



(a) Binary-tree based search



(b) Informed RRT* square search

Fig. 3: Comparison of our own obstacle avoidance algorithm against Informed RRT*. Both axis represent the x and y coordinates. Informed RRT*. Parameters: step distance $d_{step} = 0.15$, and distance to check whether target is attained: $d_{attained} = 0.25$. The start and target points are respectively in blue and green, and grey points in the Informed RRT* graph represent the resulting tree generated.

Observing the results of both search algorithms, depicted in figures 3a and 3b, our algorithm finds a path of lower cost compared to Informed RRT*. The main advantage of our algorithm is that it is deterministic and the next point search is always either towards the target, or to avoid an obstacle. On static obstacles, we have determined experimentally that our algorithm often struggles between one intermediate target and another, which is one of the main downsides of the path finding implemented. This was not a behavior that occured often during the SSL matches, and we had difficulty recreating it in a testing environment. In conclusion, while both algorithms have their positive points, the presented algorithm removes randomness, while still requiring to smooth the past in some cases. Stuttering could still be fixed by not running the algorithm at every iteration of our main software.

2.3 Optimal pass computing

Passing behaviour is one of the critical aspects of the SSL, allowing to perform more complex behaviour. In football games, players tend to pass the ball to teammates in movement, often towards the most open space. To imitate this behaviour, we have implemented a score-based search algorithm where, given a rectangular region on the field, returns the best location towards which the ball should be shot to, in a straight line (chipped kicks are not considered). The presented method was motivated by the work of team ZJUNLict, who have defined a score-based evaluation method to determine optimal passes [2]. We present a similar method but based on different conditions.

Consider a robot that currently has the ball at location p_{start}, and would like to pass it to one of its allies. Instead of selecting the best ally to perform the pass to, we search for an optimal target location instead, expecting the nearest ally robot to receive the ball. To do so, a rectangular search zone is defined on the field, and a grid of evenly spaced points is generated inside it. Each point generated defines a possible pass location, where the ball should approximately land after being shot. To select the best one, a score is computed for each point p_{pass}, based on the pass trajectory vector $\overrightarrow{v_p} = p_{pass} - p_{start}$, an enemy robot position p_e, and its linear velocity $\begin{bmatrix}\dot{x_e} & \dot{y_e}\end{bmatrix}^T$.

- **Enemy perpendicular distance to pass trajectory** For each enemy, we measure the distance to its orthogonal projection onto the pass trajectory. If the projected point lands outside the pass trajectory, the malus attributed is 0. Otherwise, equation 1 gives the malus to compute, $distance()$ being the euclidean distance function, and $proj(p, \vec{B})$ returning a point, the orthogonal projection of point p onto $\vec{B}$.

$$blocking\ score = \begin{cases} \dfrac{-1}{1 + distance(p_{start}, proj(p_e, \overrightarrow{v_p}))} & \text{if } d \leq 1.5 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

  The total sum computed for each enemy robot is a malus to the final score, being higher when the distance is smaller. After a certain threshold, distance is considered to be too high to be a danger for this pass trajectory, so the malus becomes 0.

- **Closest ally to pass location** Naturally, the closer the ally is to the expected landing location, the less time is required to catch the ball and manipulate it further. Due to electromagnetic interferences coming from our kicker mechanism, the maximum kicking power is limited, thus we chose to not take into account outgoing ball speed when calculating this score. By computing the closest ally's distance using d_{min} as defined in (2) the score is given by equation 3, with P_{allies} denoting the set of (x, y) ally positions on the field.

$$d_{min} = \min_{p_a \in P_{allies}} distance(p_a, p_{pass}) \tag{2}$$

$$proximity\ score = \begin{cases} \dfrac{1}{1 + d_{min}} & \text{if } d_{min} < 1.25 \\ -2 & \text{otherwise} \end{cases} \tag{3}$$

The malus applied if distance is too small is to discourage short passes.

- **Coordinate x position on the field** Passes must allow progress on the field. As arriving next to a goal line is not ideal, the maximum score value is attained at the middle of half of the touch line. Depending on whether the playing team is on the positive or negative half of the field, equation (4) give the respective score depending on the side.

$$field\ progress\ score = \begin{cases} -\dfrac{x(x + \frac{9}{2})}{5} & \text{if on positive half.} \\ -\dfrac{x(x - \frac{9}{2})}{5} & \text{otherwise} \end{cases} \tag{4}$$

- **Enemy time of collison to the pass trajectory** Based ony an enemy robot's velocity vector, we perform an iterative search over the trajectory to compute possible collisions with each enemy robot. Tigers Manheim have described their algorithm for collision checking in [4], section 3.3. A similar approach has been implemented, attributing an increasing collision circle for a robot over time.
  We iterate on the trajectory at fixed time steps. At each step, we compute the collision circle of an enemy robot and check for collision. Origin of this collision circle is biased using the enemy's velocity vector, as robots rarely change abruptly their driving direction during a match.
  Because the collision circle is a prediction, the starting score at t_0 is set to 1. This value decreases as the current time step increases. If a collision is found at time step t in seconds, equation 5 gives the predicted collision score.

$$predict\ collision : f(t) = -e^{-2*t} \tag{5}$$

  Computing this score requires to predict the ball's position over time after kicking. For the moment, we use an approximation to test this algorithm. In the future, we plan to create a kinematic model for the ball based on our mechanical kicker structure.

By summing up these score computations, alongside a given weight for each score function, total is compured is computed by $\sum \omega_i s_i$ with ω_i the weight for each s_i score computed. This allows us to tune the algorithm to favour certain conditions over others. Figure 5 presents the resulting score heatmap obtained on the world configuration depicted by figure 4.

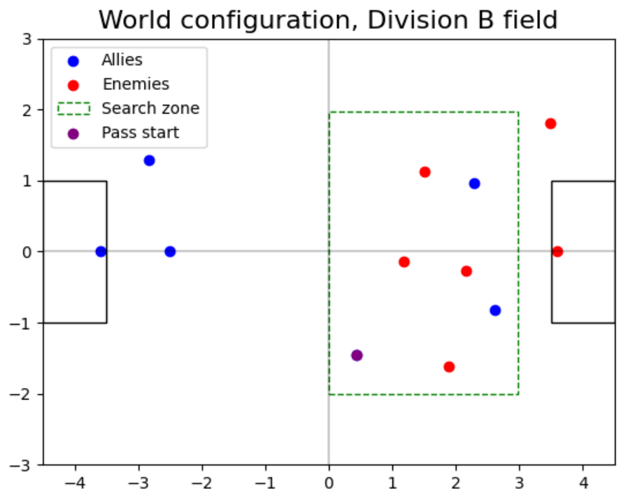


Fig. 4: Testing world configuration for the pass score estimation algorithm implemented. The red points are the enemies and is placed on the positive half of the field. Search grid has been limited inside the enemy's side. Allies are depicted in blue.

2.4 Refactor of game state detection

During last competition, many fouls were committed by our robots due to a misunderstanding of the rulebook. The error could not be fixed easily because the current codebase was parsing the current game state using a combination of conditions and "j" statements. Thus, we have rewritten the state machine detection from scratch to improve maintenability and readability of the code. Since our AI codebase is in Rust, we first searched for librairies to implement the game states of a match. But because it is required to handle rollback in the state machine, we have decided to not use any library, to keep the codebase simple, as we do not need all of the functionalities of a finite state machine. In particular, the state machine of a match runs infinitely (thus no accepting state).

The Rust language provides a match statement, which allows for pattern matching in the code. Using this mechanism, most of the state machine detection is handled inside a single transition function, to correctly parse the game states of the match. The transition function performs pattern matching over four variables: current game state, latest command sent by the referee, reference poistion for the ball and latest game event. With unit testing, this has proven sufficient to handle game states correctly, and is much readable to the human eye, as pattern matching looks closely to listing the transitions of the state machine on paper.

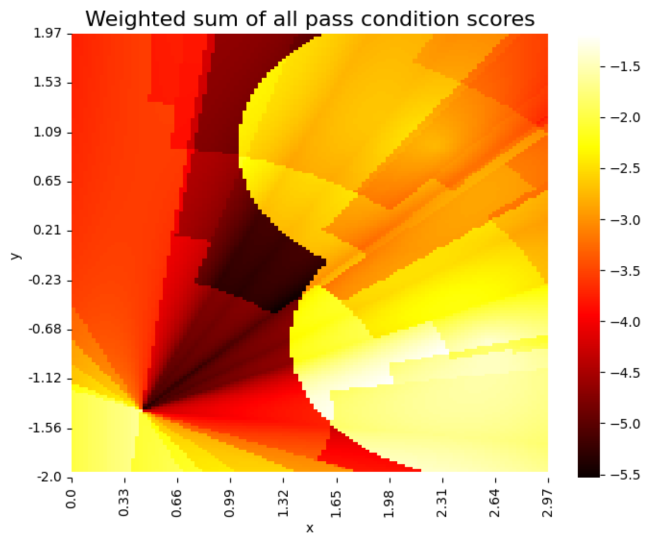


Fig. 5: Resulting weighted sum for each point computed inside the search space. The heatmap is not smoothed and presents a value for each point. All weights of the score functions were set to one, with the exception of the "closest ally" condition, whose weight was set to 4.

Note that the transition function is not called at every iteration of our complete system. It is only called whenever a new command from the referee has been sent, or if the current game state dynamic, meaning it depends on time or on another condition (such as the "FreeKick" state, that transitions to "Running" once a robot touches the ball).

3 Firmware improvements

Last year, most of our firmware had been rewritten in Rust. The last part missing is code for motor control. As the team is discussing whether to use a Rust-only communication protocol, having a stable version written in C++ is the current objective until the Rust version is ready for a full transition. By doing this, it will be beneficial to perform comparisons between the two firmwares, and leaves us with a stable solution if the Rust version were to have problems.

3.1 Receiving feedback from robots

The team currently uses the nRF24L01+ radio module to communicate with the robots. Up until now, command sending was single way, we didn't receive feedback from the robots. This year, we use the payload ACK capability of the radio module. Each time a command is sent to a robot, it responds with an acknowledgement packet to confirm reception. The radio module allows us to put a 32 bytes of data inside the ACK packet, thus enabling full-duplex communication, allowing us to receive ball detection data and motor speeds.

3.2 Collision-free communication

Upon arriving at the competition during RoboCup 2024, we fixed our radio communication which emitted packets that seemed corrupted to the robots. This year, the communication has been made more robust.

As mentioned in our previous TDP, the original goal to enable reception of feedbacks while not causing any pakcet collision was to use Time Distributed Multiple Access (TDMA). Team TIGERs Manheim have described such an use [5] to be viable for collision-less communication. The idea behind TDMA is assigning a fixed slot time for communication, such that packets for robot 0 are always sent in this time window. Currently, speed control of our robots is done by continuously sending a packet that contains the target speed for the robot to attain. So instead of implementing TDMA, we have chosen to simply send packets sequentially, with the capabilities of the nRF24L01+ module.

The radio module raises an interrupt pin on correct packet emission. Since auto-acknowledgement is enabled, as mentioned in the previous section, once this acknowledgement has been received by our emitting station, or if the acknowledgement times out, the interrupt pin is raised. After receiving the interrupt, we load and send the packet for the next robot, and immediately send it. Because each packet is sent sequentially, this effectively creates a unique slot time for each robot. This approach is very close to TDMA and is possible thanks to the payload ACK feature of the nRF24L01+.

4 Acknowledgements

We would like to thank the region of Nouvelle-Aquitaine for providing funds thus far, the University of Bordeaux, the Bordeaux Institute of Technology and. Thank you to the previous team members for their prior work and to the CATIE for their continuous collaboration. Finally, thank you to our sponsors ofor their support.

References

1. P. Felix, O.Ly G. Passault, E. Schmitz, S. Loty, C.Laigle, A.Chauvel, T W.Menier, V. Chaud, L. Paille, E.Miqueu, B.Chew, J.Gautier, T.Decabrat, R.Denieport, and P.M.Ancele. *NAMeC - Team Description Paper Small Size League RoboCup 2023 Application of Qualification in Division B*, 2023.
2. Zheyuan Huang, Haodong Zhang, Dashun Guo, Shenhan Jia, Xianze Fang, Zexi Chen, Yunkai Wang, Peng Hu, Licheng Wen, Lingyun Chen, Zhengxi Li, and Rong Xiong. *ZJUNlict Extended Team Description Paper for Robocup 2020*, 2020.

3. Daichi Miyajima, Kosei Naito, Hayato Mitsuda, Kazuaki Harada, Mizuki Nonoyama, Ryo Shirai, Futa Sato, Ryuto Tanaka, Yota Dori, and Toko Sugiura. *KIKS Extended Team Description for RoboCup 2023*, 2023.
4. Nicolai Ommer, André Ryll, Michael Ratzel, and Mark Geiger. *TIGERs Mannheim Extended Team Description for RoboCup 2024*, 2024.
5. Andre Ryll and Sabolc Jut. *TIGERs Mannheim - Extended Team Description for RoboCup 2020*, 2020.