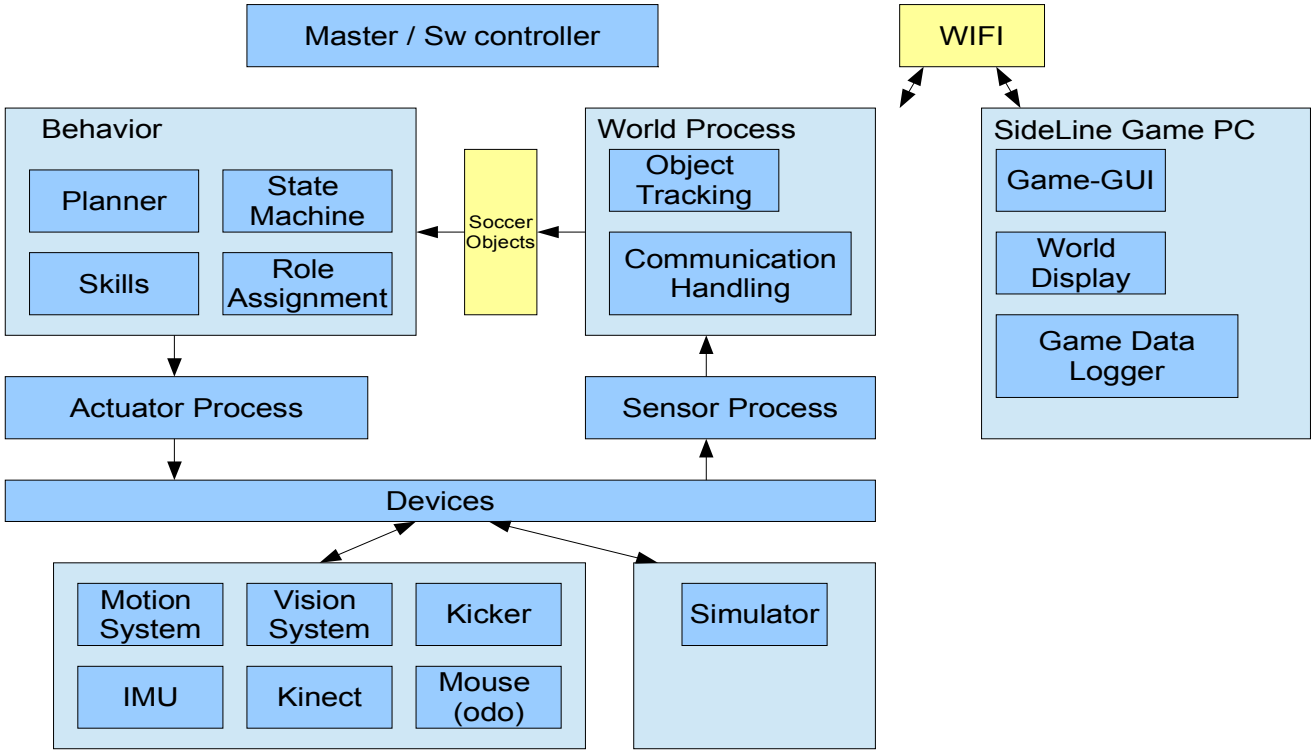
Robotsports Software Details

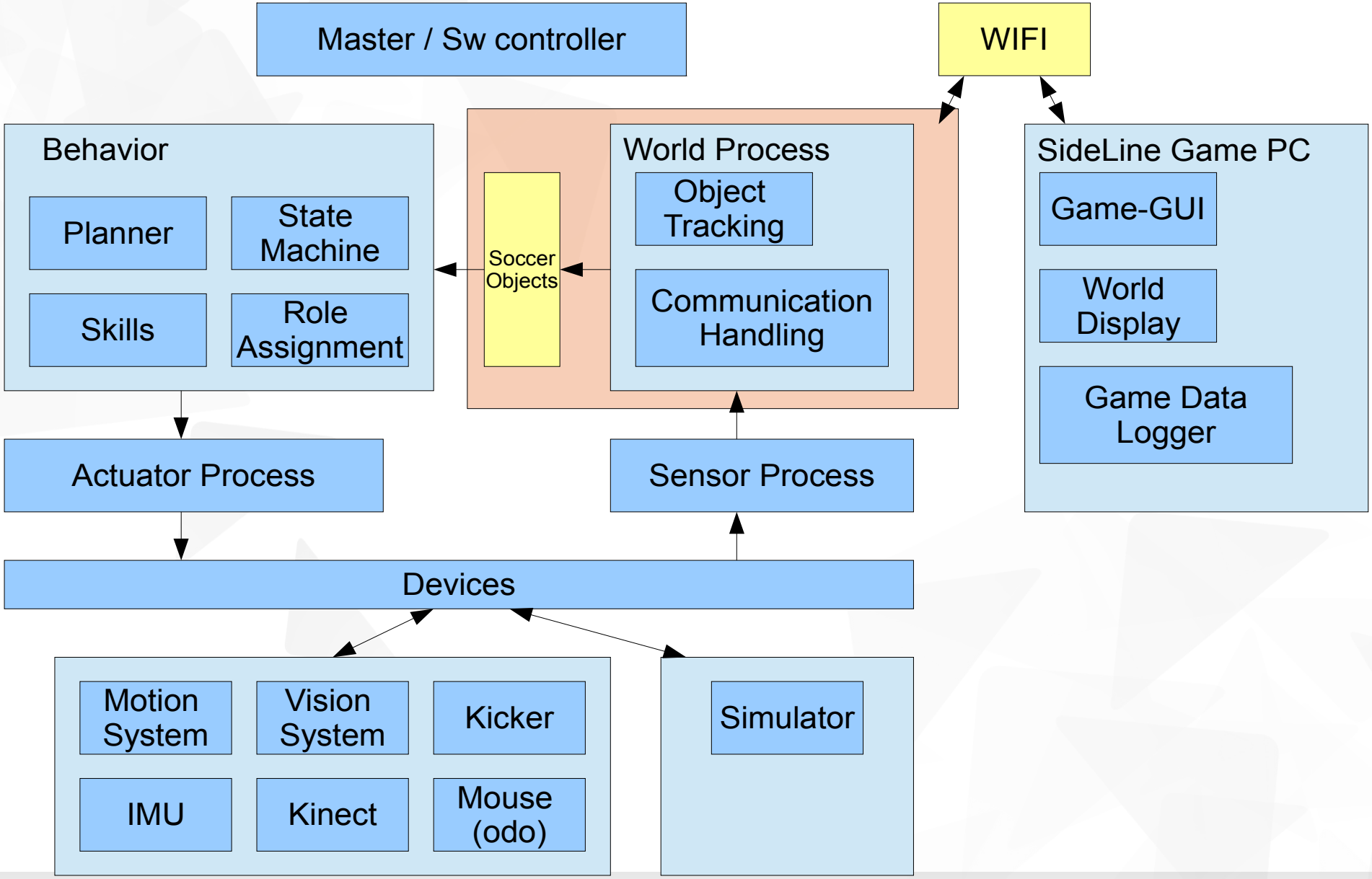
Contents

SW architecture overview

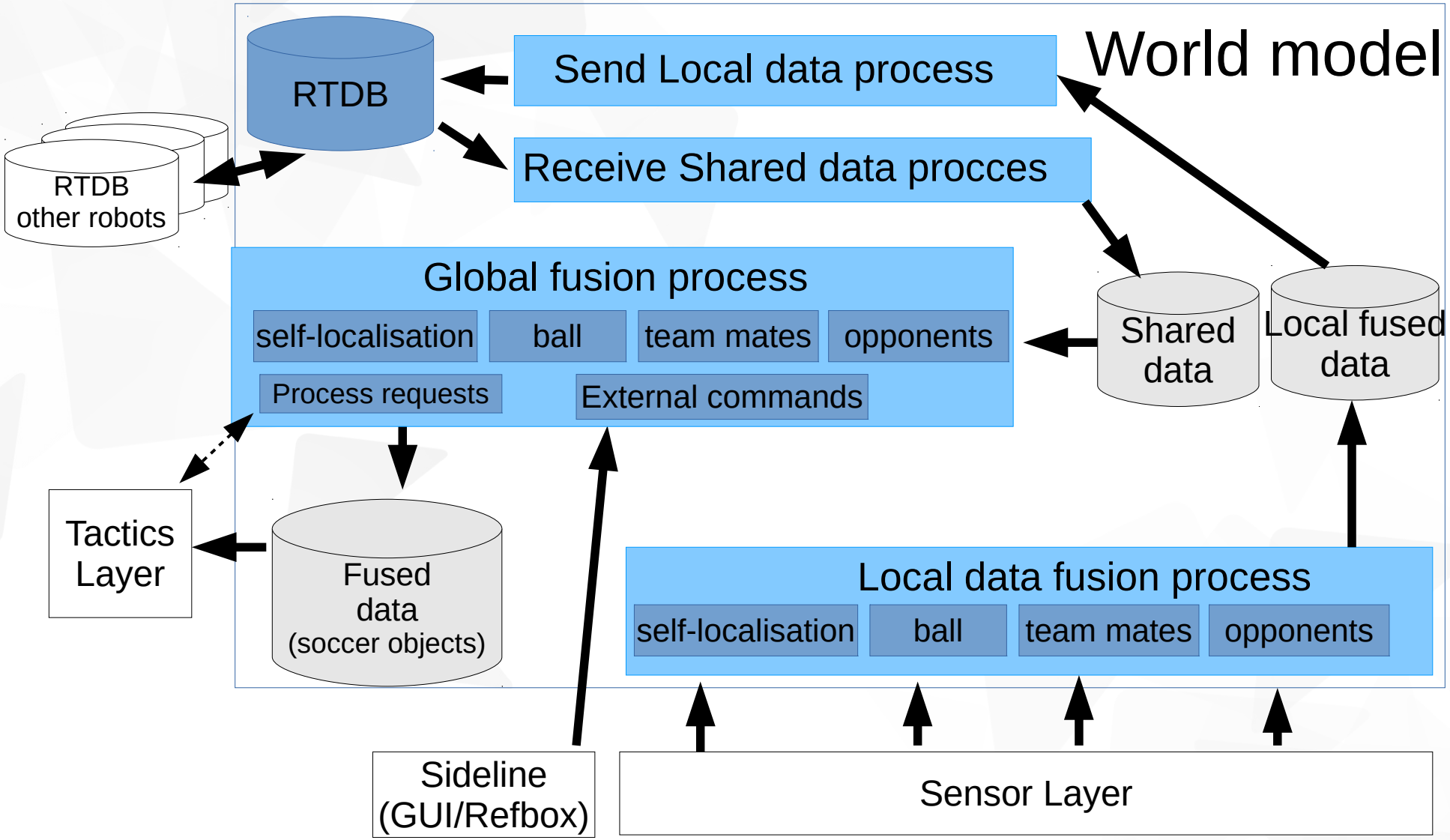


Robot sport: World model

SW architecture overview



World model overview



World model Processes

- Local data fusion process: combines local sensed data.
- Send local data process: send the local fused data to other players
- Receive shared data process: collect data from other robots
- Global data fusion process: combine all global data and local fused data to soccer objects (for tactis)

Software Interfaces with World model

- Tactics
  - soccer objects and its properties (ball, players, field)
  - Requests for enable/disable generic calculations
  - Requests for time-outs (start and trigger when expired)
- Real-time data base (RTDB): network synchronized
  - Synchronize data (soccer objects) with other robots
- Sensors
  - Read data from sensors
- Sideline:
  - Refbox-listener: receive commands of referee (via refbox)
  - GUI: commands / request for info of operator.

Self-localisation

- Purpose: It tries to keep estimate the position of the player in the field. A lot of sensor data can be used for this estimation.
- Possible used data is:
  - Static data like: thickness of the white field lines, the dimensions of the field, the original starting position of the robot.
  - Dynamic data like: estimated fieldposition from vision, odometry, postion of other robots, position of the observed goals.

Soccer Objects

- Purpose: the WorldModel will eliminate as much as possible positions for an object.
  - Example: in case of a ball multiple positions can be communicated via RTDB or seen by vsion. Currently the ball with the lowest uncertaintity will be selected as the ball for tactics. In the future it can decide to go for a more defensive ball in certain situations.
- The soccer objects:
  - Me, Team, Opponents, Game, Field

World model process – some steps

- Match new measurements with the previous measurements is one of the first steps in the worldModel process (object corresponding).
- Position tracking of dynamic objects like the ball, players and goals is a reponsibility of the WorldModel. Herefore several sensors can be used. Vision data, but also datafrom the teammates (via RTDB). Physical limits like a maximum speed can also limit the possible positions for an object.
- The Refbox communicates the game state, which is handled the worldModel.
- The GUI communicates commands and requests, which are handled the worldModel.

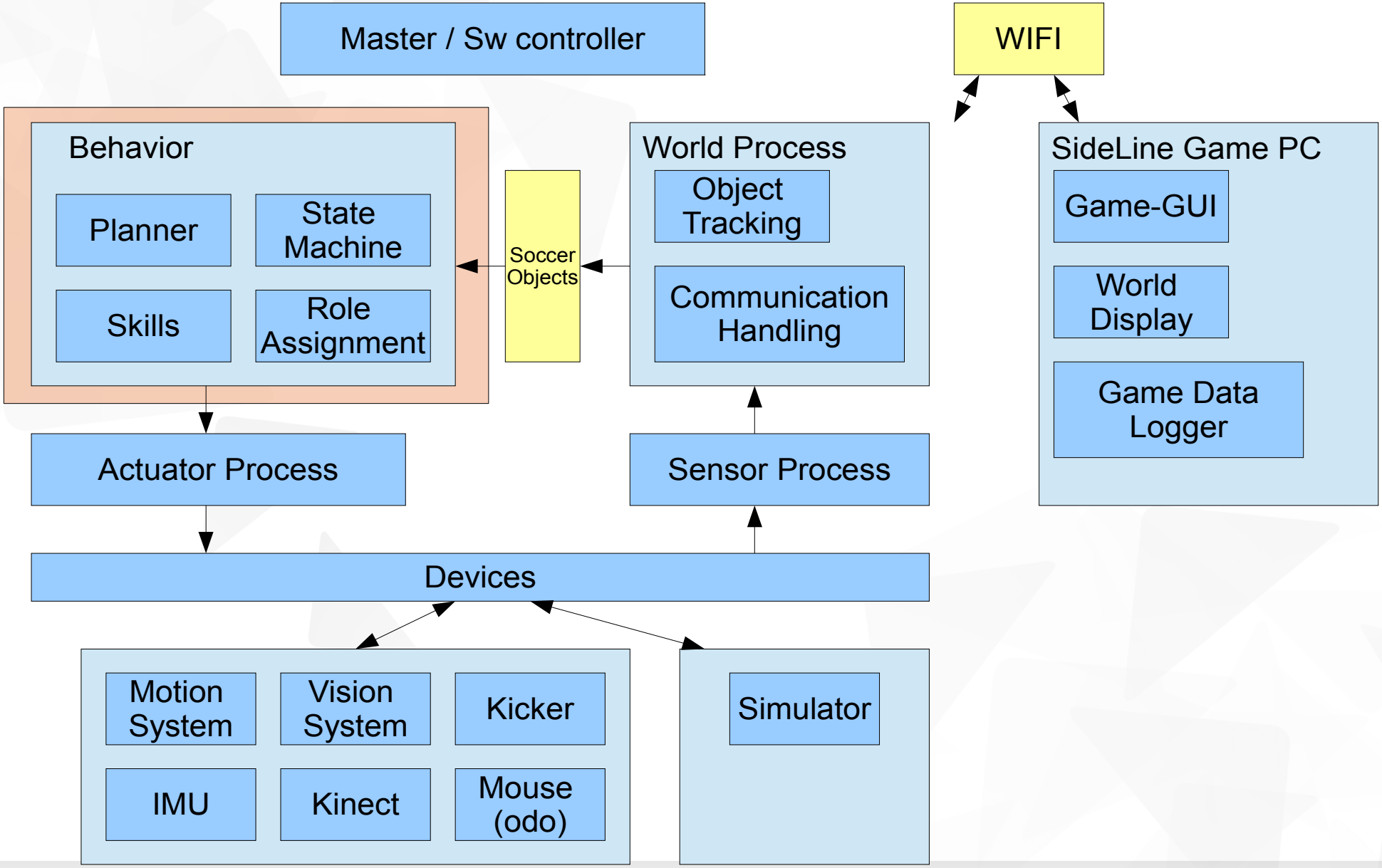
Coordinate systems

- There are three coordinate systems used in a robot
  - Field / Global: absolute position in the field.
    (origin in center of field, +Y is towards opponent goal)
    - This coordinate system is redefined by self-localisation
  - Relative / Local: positions relative to robot.
    (origin in center of robot, +Y is towards front of the robot)
  - Continues / Extrapolated: a continues non-corrected coordinate system (used by sensor to avoid discontinuities)
    Only be used for special situations
    (par example: ball movement detection)

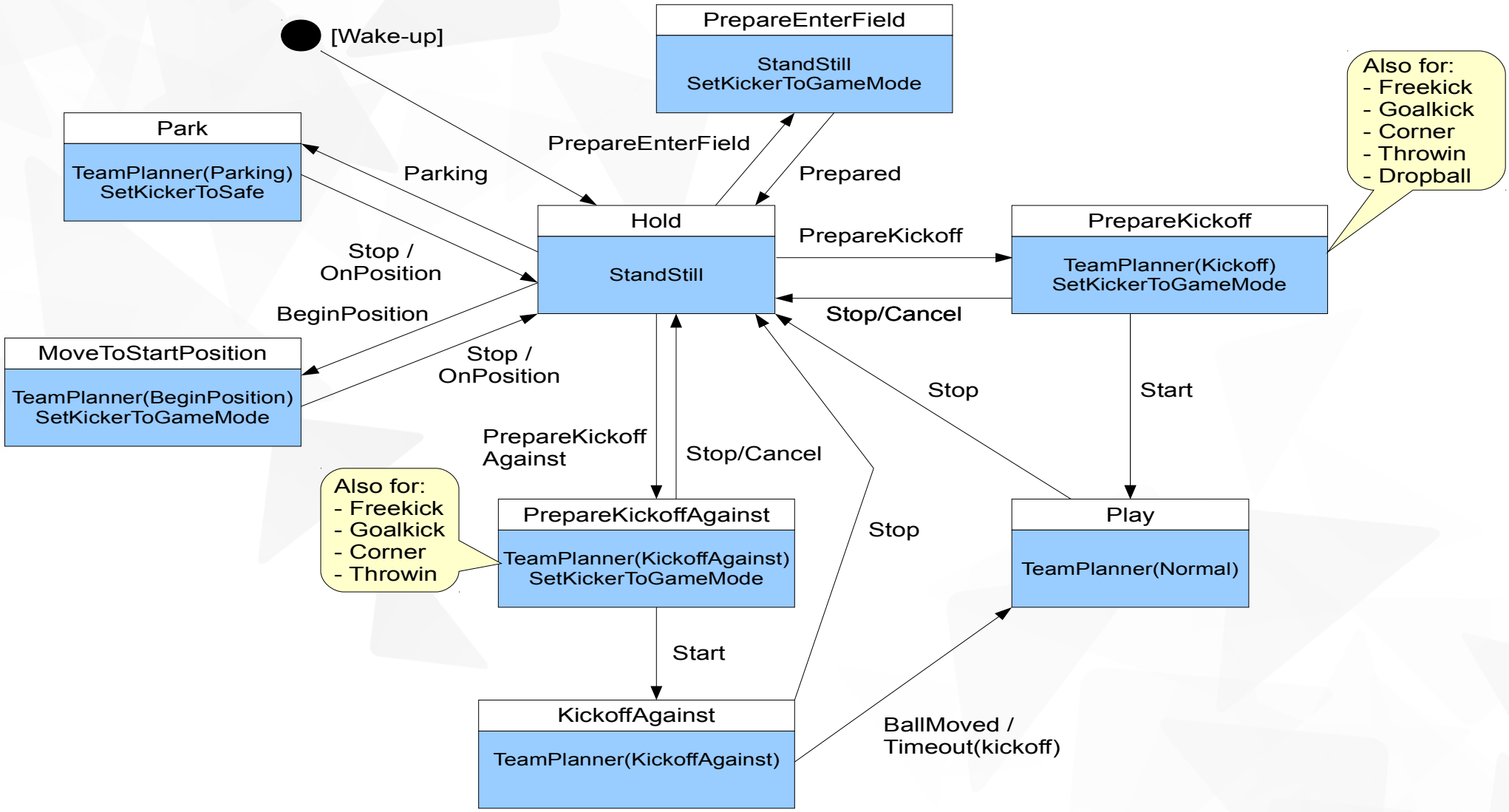
  - Most data will in be local CS or in field CS.

Robot sport: Behavior Software

SW architecture overview



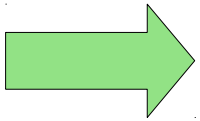
Robot Sport Game states
(Handling refbox-states via Stactics)



Statics compiler: .stx => .stl

.stx-file (partial)

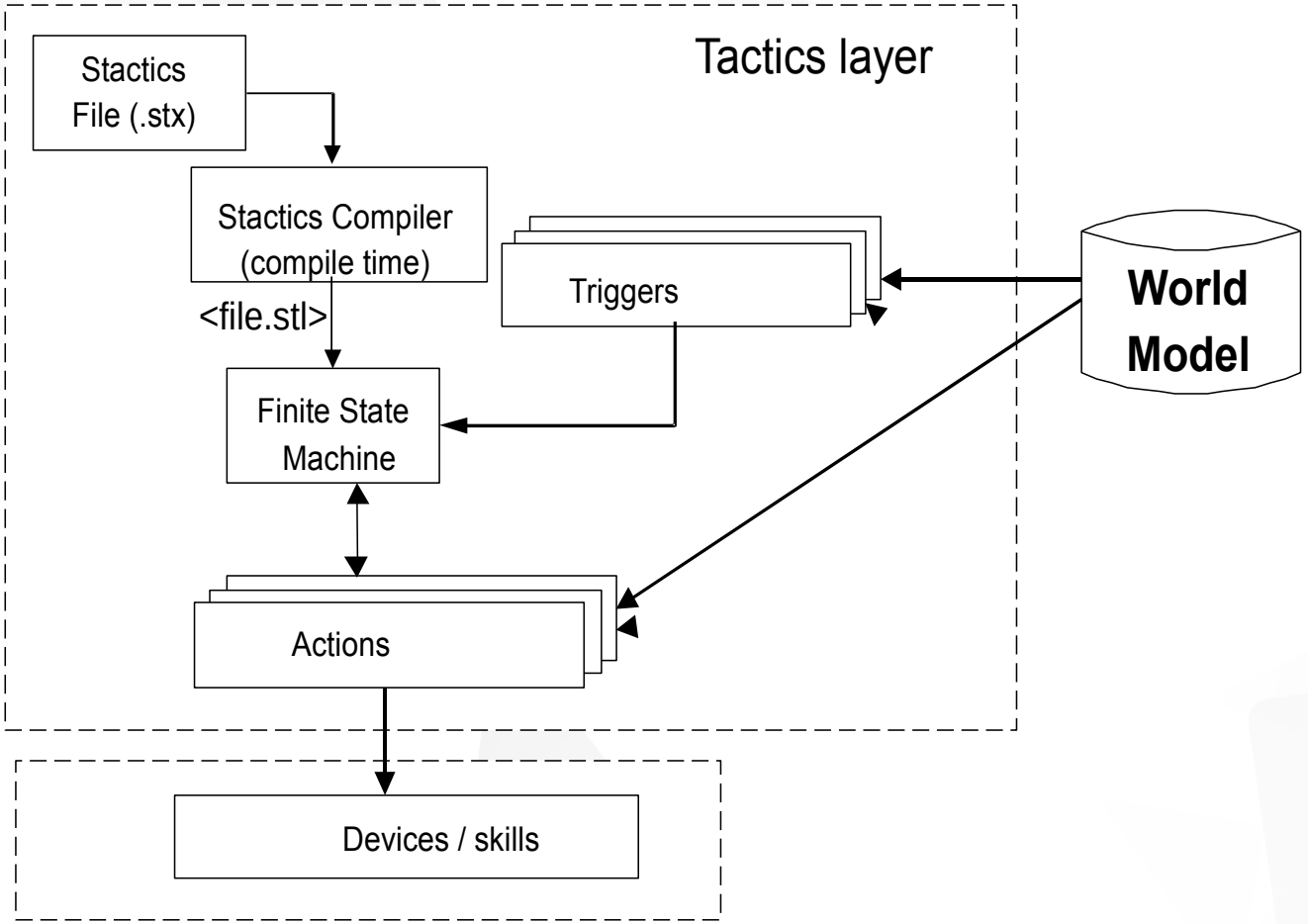
```
State Player
  Sub_state =
  {
    Hold,
    Play
  }
  Decision_skill =
  {
    CommandReceived(Stop) -> Hold
  }
End_state

// Starting State
State Hold
  Action_skill =
  {
    StandStill
  }
  Decision_skill =
  {
    CommandReceived(Start) -> Play
  }
End_state
```



.stl-file

```
<Stactic_file>
    <Parser_version>
        <number>4.2</number>
        <date>2008-08-18</date>
    </Parser_version>
    <Stactic_info>
        <file_name>demo1.stl</file_name>
        <generation_date/>
        <md5sum>...</md5sum>
    </Stactic_info>
    <State_names>
1 Player.Hold
2 Player.Play.FindBall
3 Player.Play.FollowBallOnSpot
    </State_names>
    <State_descriptions>
1 C CommandReceived ( Stop ) 1
1 C StandStill ( )
1 C CommandReceived ( Start ) 2
2 C CommandReceived ( Stop ) 1
2 C TurnAround ( )
2 C DoesRobotSee ( ball ) 3
3 C CommandReceived ( Stop ) 1
3 C TurnToBall ( )
3 C DoesRobotSee ( no_ball ) 2
    </State_descriptions>
</Stactic_file>
```

Stactics - blocks



Stx file:
Soccer state behavior syntax.

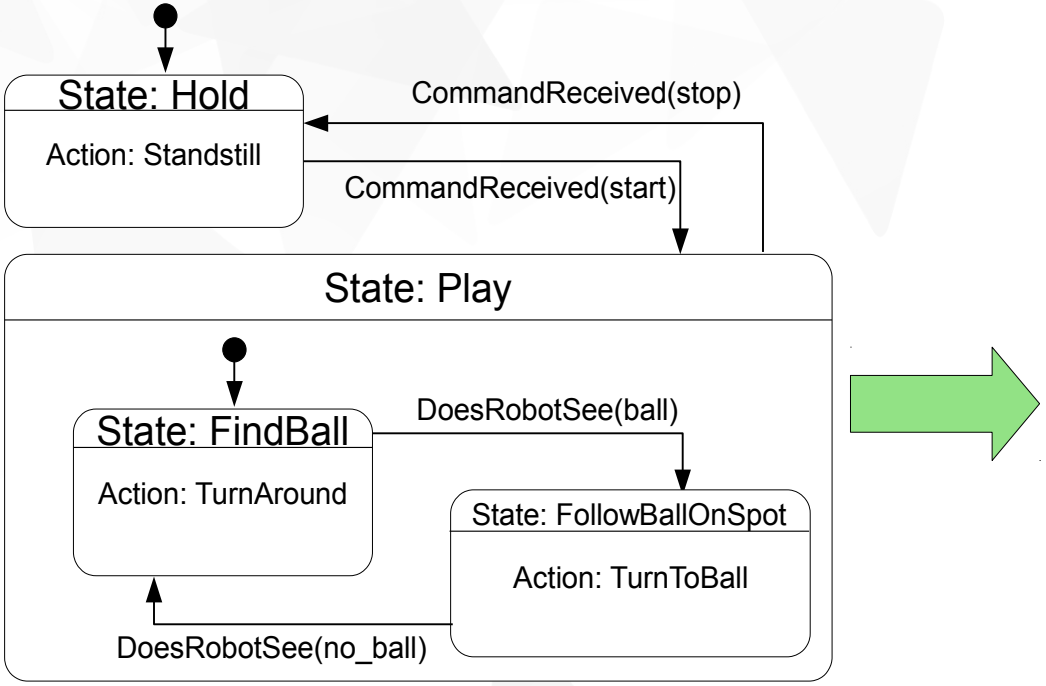
Stactics Compiler
The compiler translates the .stx into a machine state list (extension .stl) for the Finite State Machine.

Finite State Machine (FSM)
FSM controls the triggers and active actions. input the .stl-file

Triggers
Evaluate criteria on request by FSM, using information from the worldmodel.

Actions
Behavior of the robot, which controls (de)activation of robot sports skills.

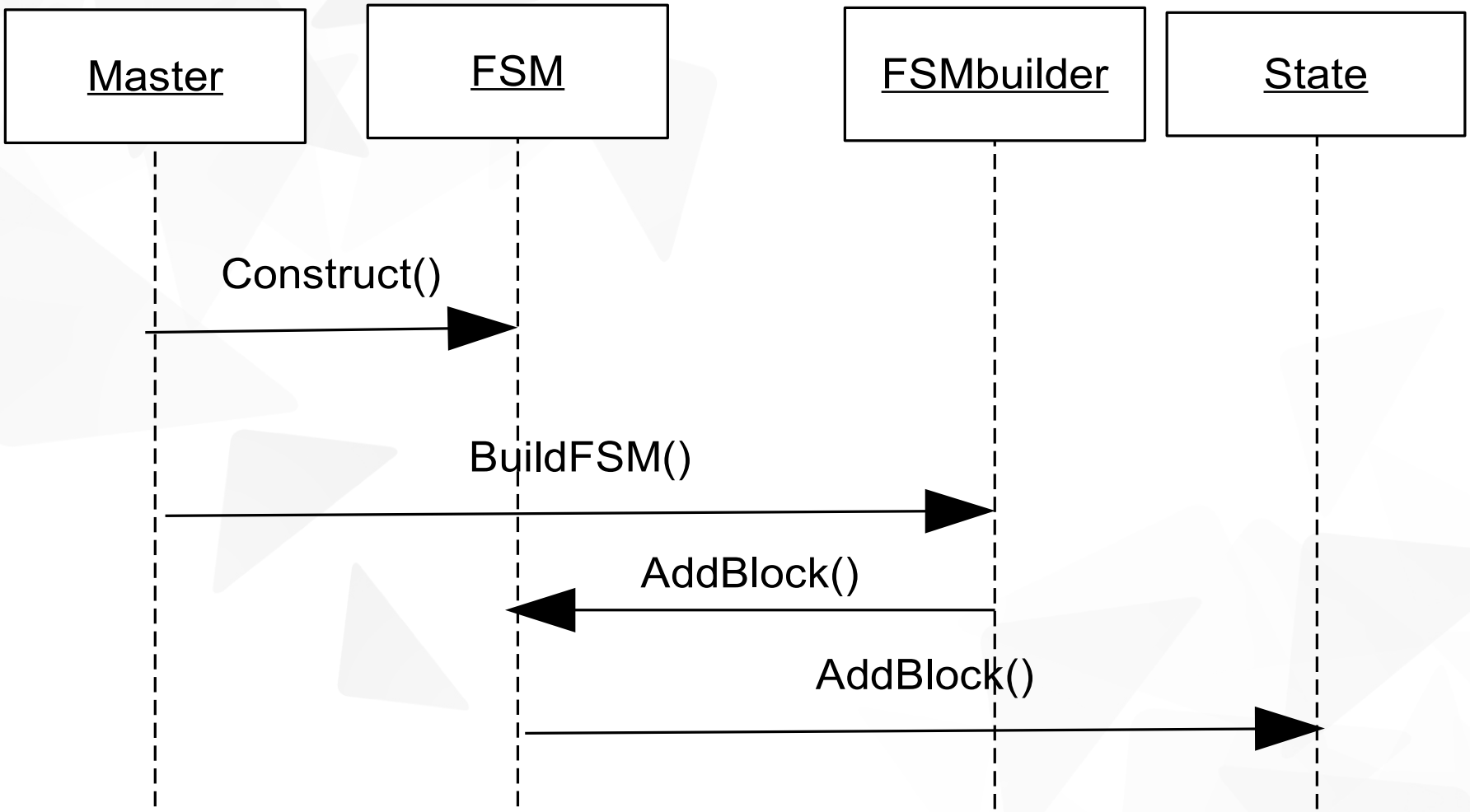
Stactics: idea (states) to .stx



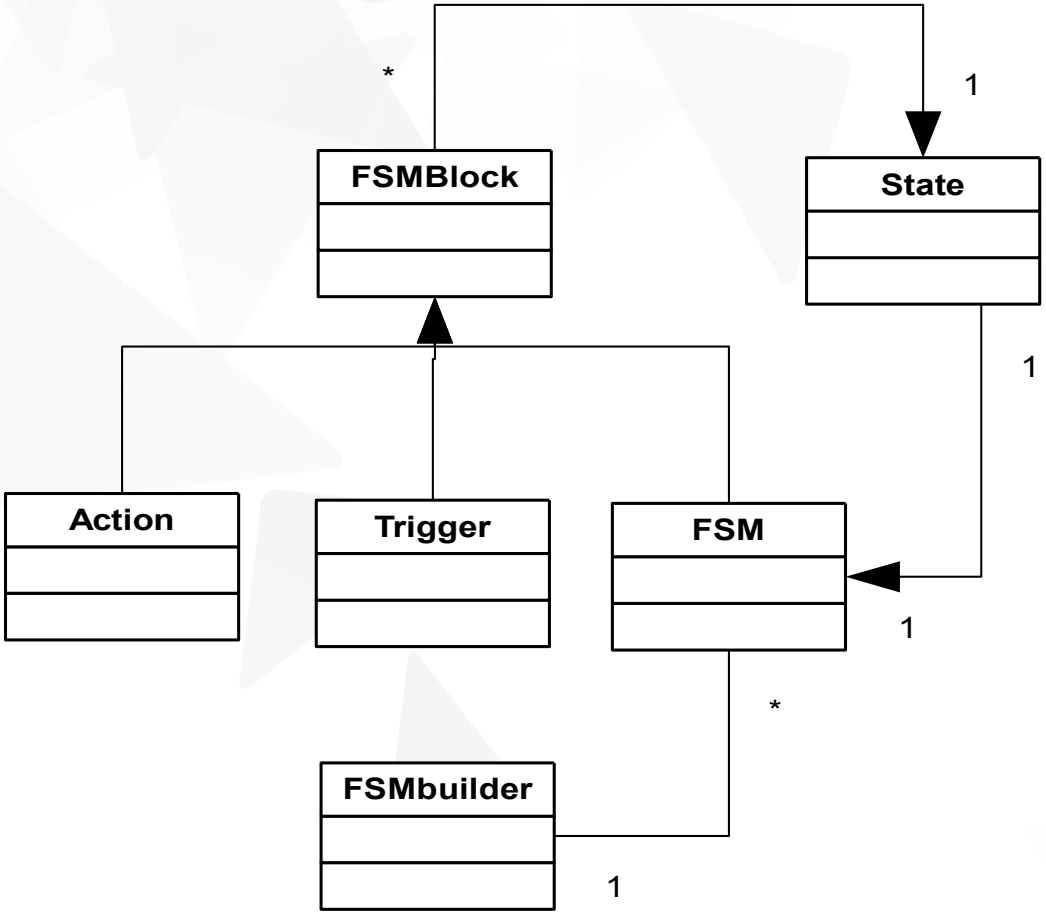
.stx-file (partial)

```
State Player
  Sub_state =
  {
    Hold,
    Play
  }
  Decision_skill =
  {
    CommandReceived(Stop) -> Hold
  }
End_state

// Starting State
State Hold
  Action_skill =
  {
    StandStill
  }
  Decision_skill =
  {
    CommandReceived(Start) -> Play
  }
End_state
```

Dynamic FSM built-up during init (input .stl) 1/2



Dynamic FSM built-up during init (input .stl) 2/2



Team behavior (situation based)

- **Pre-condition:** usage of the synchronized real-time database (RTDB developed by Cambada); all robots will have a clear view on the positions of own team and the ball (and hopefully also the opponents).

- **Team-strategy design decision:**
  All players will have the same data as input for the behavior, therefor all robots can make their own decision what there role (and target position) is based on the situation. No explicit communication or central coordinator is required

Layers in role assignment and planning

- **Team-planner:** Assign all roles (start with most important role, than the next important role), only goalie role will be fixed. The other roles will be dynamic and based on the field situation.
- **Robot-planner:** calculate paths (return path-costs) for a given field-situation and game-situation (objective). Output rough X-Y path.
- **Smoother**: Smooth the give rough X-Y path, taking max. velocity, acceleration and robot positions into account. Result: smoothed path, with velocity and Rz
- **Path-executor** (skill): Execute the smoothed path with the robot.

Team-planner: functionality

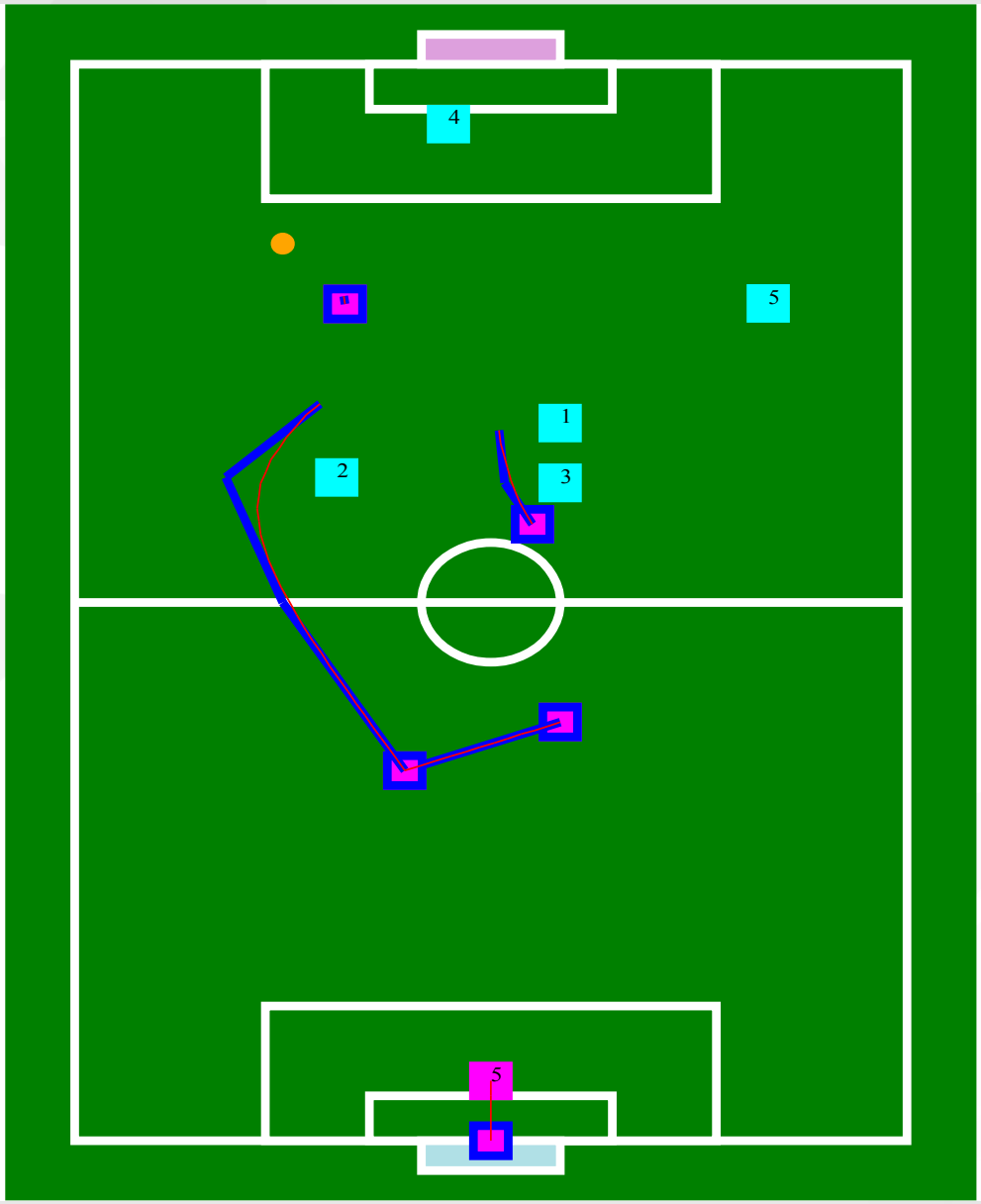
- Calculate path (via robot-planner) for all field players the most important role (often move to/with ball). The costs of the paths (length and some penalties) can be ranked, cheapest cost will get the role. Player with the role is assigned, the remaining players will be used for assigning the next role, till all roles are assigned (and the rough path is already calculated).

  Example on next slide

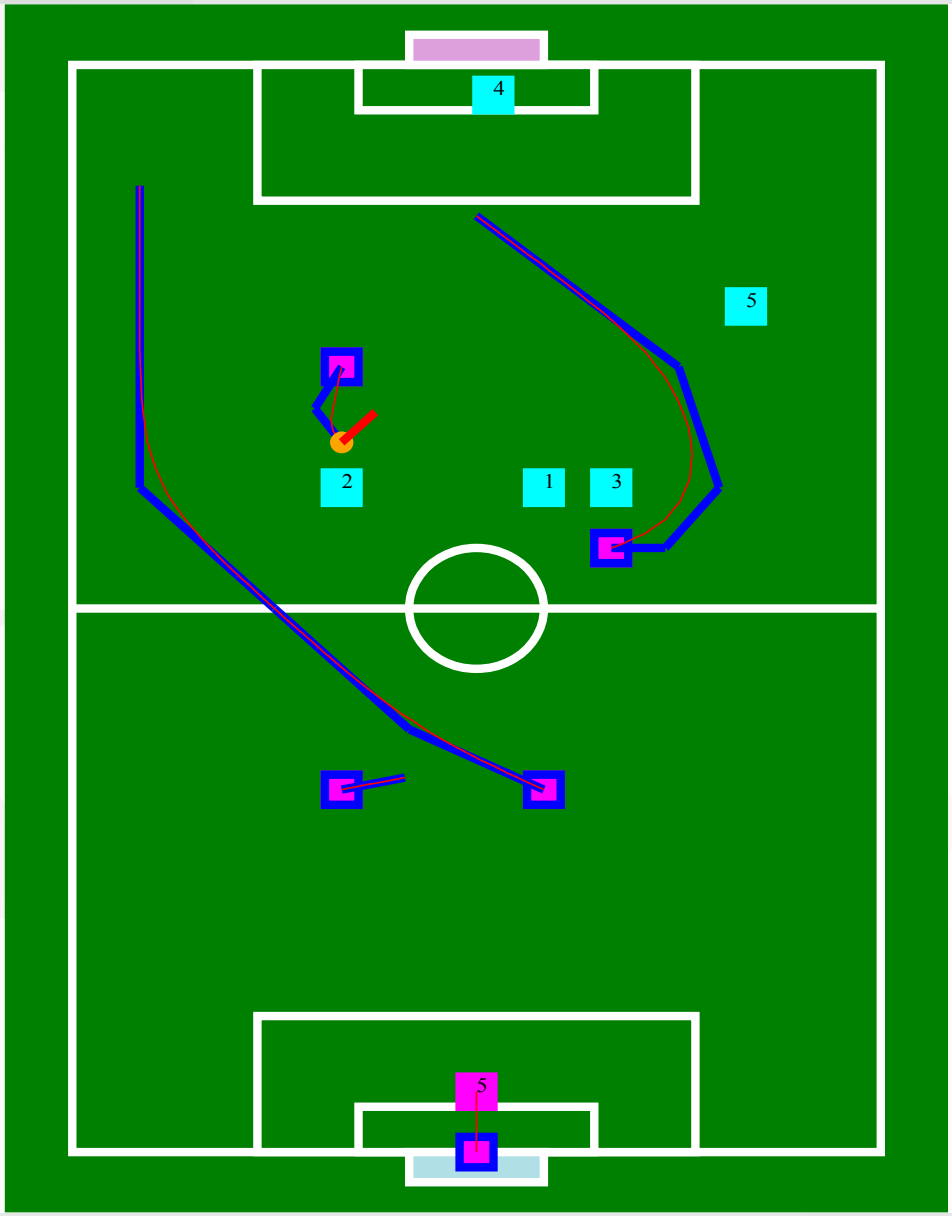
Team-planner: Example of defending behavior

- Assign: goalie (fixed role)
- Assign robot to intercept ball
  - Calculate path to ball for 4 players with robot-planner. Robot with cheapest path gets the role. (X-Y path is known)
- Assign sweeper role (optional)
  - Calculate path to sweeper position for remaining 3 players.
- Assign most important defense position
  - Calculate path to defensing position for remaining 2 players.
- Assign most important defense position
  - Calculate path to defensing position for last player.

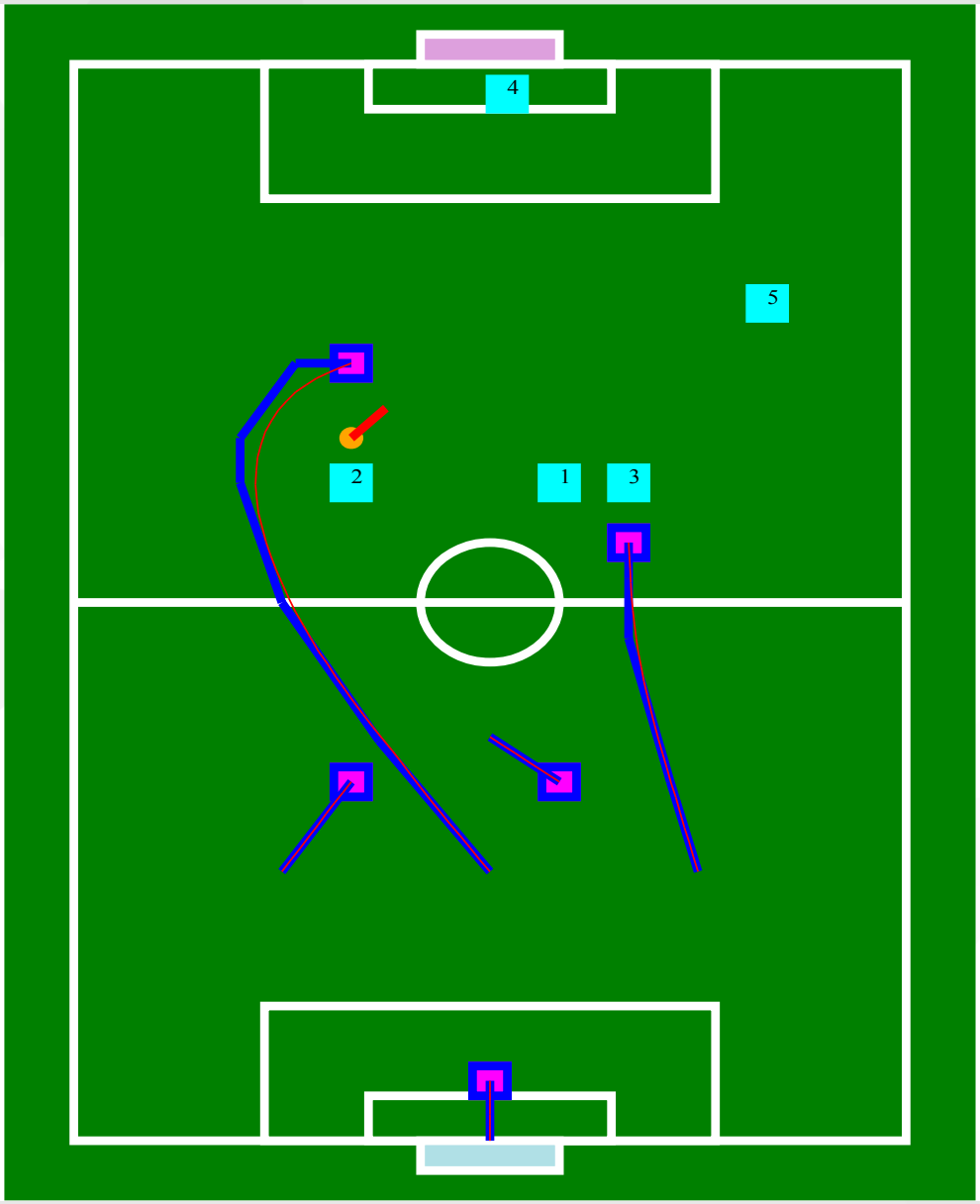
Team planner: example Goal kick against



Team-player: example Free-kick



Team-planner: example Start positions



Robot-Planner: Rough X/Y planner 1/4

- Calculates a rough path (only X-Y)
- Calculation is quite fast
- A* is used to find shortest path algorithm
- Smart random points are added for better choices

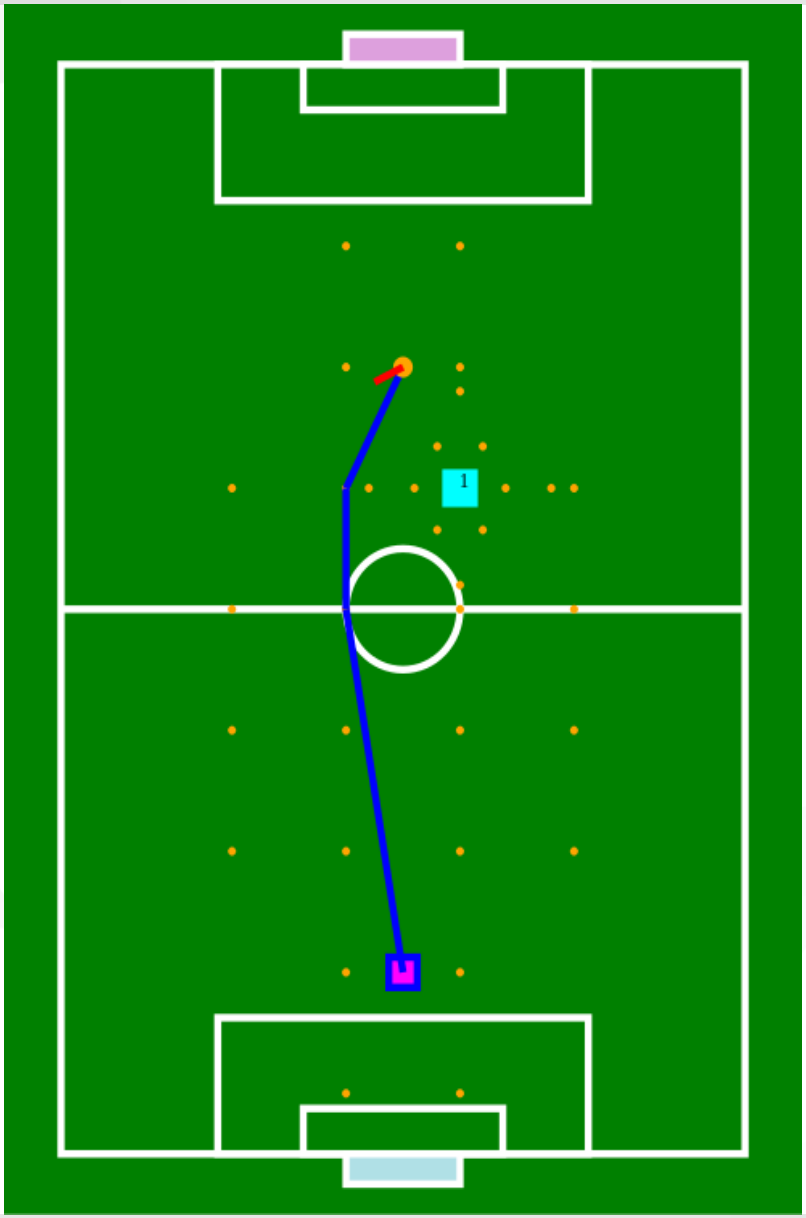
Robot-Planner: Rough X/Y planner 2/4

- Robot-planner is inspired on a Visibility graph
  - Visibility graph was first applied on Shakey the Robot, the first general purpose mobile robot to able to reason about its own actions (1969)
  - A visibility graph is a graph of intervisible locations, typically for a set of points and obstacles int the Euclidian plane. Eache node in the graph represents a point location, and each edge represents a visible connection between them. That is, if the line segment connecting two locations does not pass through any obstacle, an edge is drawn between them in the graph.

Robot-planner: features 3/4

- Moving target (ball) is taken into account
- Alternative target-position and path in case wanted position is unreachable or ball is going outside the field
- Smart random points are added for better choices
- Limitations on moving target
- No bounce or intercepting is taken into account
- Assume constant velocity of target
- Interception point is simple trigonometry

Example: robot planner to moving ball



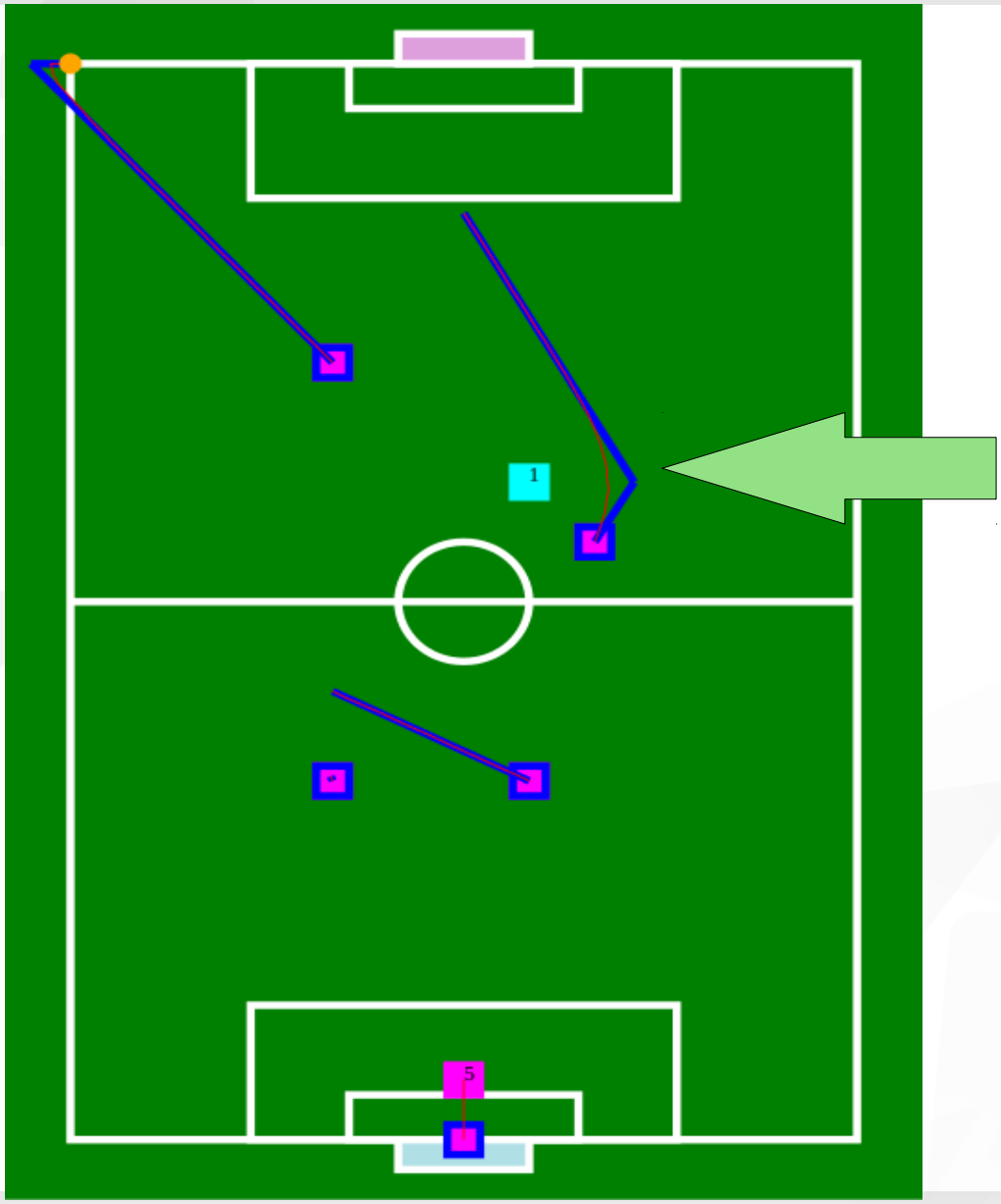
Robot-planner: future extensions / changes

- Increasing the field size should not be a problem
- 22 players and a few referees in the field will not lead to unmanagable calculation times.
- If calculation time is an issue, replanning can be performed on small deltas with D* Lite algorithm

Path Smoother

- Input:
  - path from the rough X/Y path from robot-planner
  - Target information: move with ball, to ball or other position
- Output: smoothed path (X/Y/Rz)
  - Taking maximum velocity, acceleration into account (different parameters for dribble or dash)
  - Add Rz positions to path

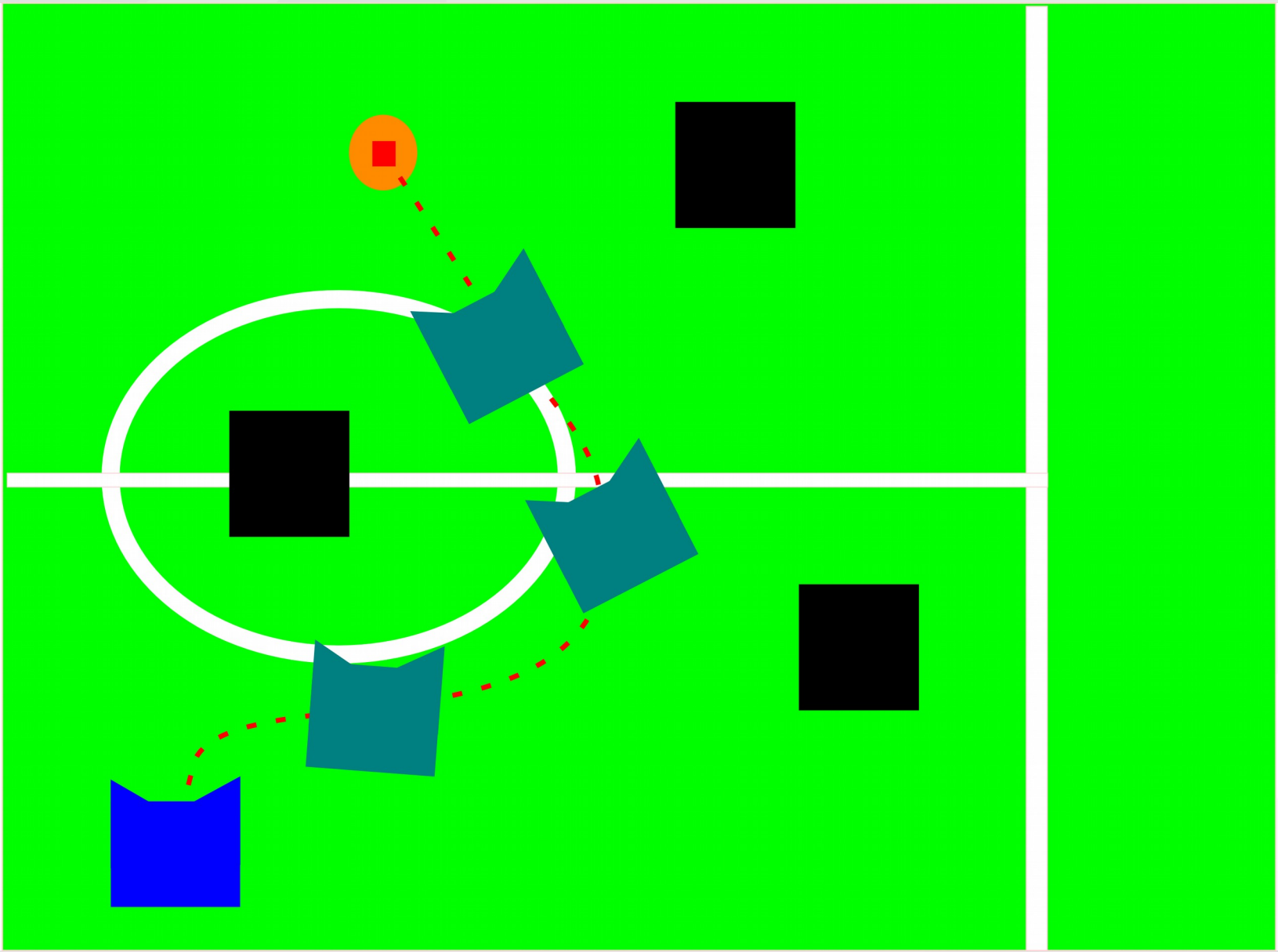
Path Smoother: example result



Path smoother: functional breakdown

- Rough X/Y path is split up in smaller parts (mini-vectors)
- Smoother re-organizes the mini-vectors taking objects into account and keep the ball inside. (multiple iterations possible)
- Only a part of the path is smoothed (calculation time improvement)
- Different Rz for move to ball or move with ball (next slides)

Smoother: path to ball



Smoother: Path with ball