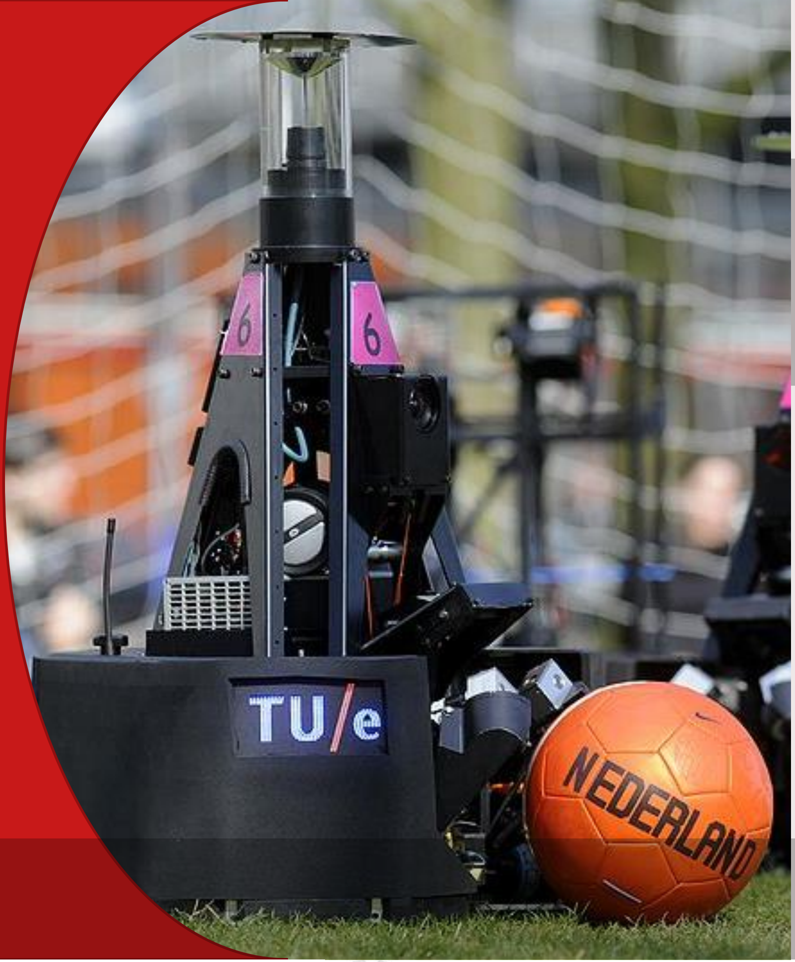
Spatial representation of skill feasibility for decision making

APPLIED TO ROBOTIC FOOTBALL

Committee
René van de Molengraft
Ömür Arslan
Elena Torta
Peter van Lith

Coach
Hao Liang Chen

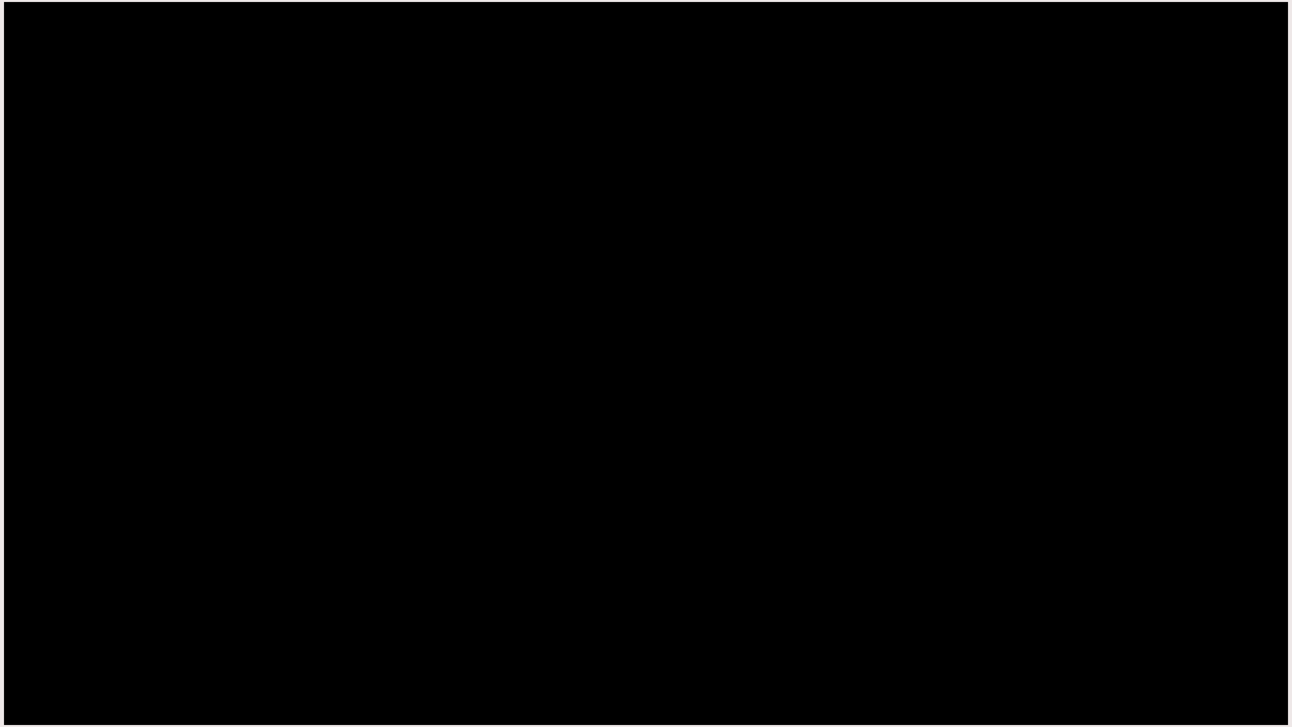


J.P. van der Stoel, Thesis

Mechanical Engineering, Control System Technology

Tech United football team world champion 2022

Turtle



Content

1. Introduction and problem definition
    - Research question
    - Current decision making
    - Problem definition
2. Semantic representation of skill feasibility
3. Building the semantic map for robotic football
4. Implementing in the Turtle software
5. Results
6. Conclusion

“Could the **strategy** of robotic football benefit from **semantic knowledge** in in the **world model**?”

- **Strategy**: a general plan to achieve one or more long-term or overall goals under conditions of uncertainty. *(wikipedia)*
- **Semantic knowledge**: giving ‘meaning’ to information → concerning strategy.
- **World model:** the robot's inner map of the environment (2D).

Multi Robot Task Allocation and Decision Making

***MRTA (STP)**: how to divide the effort (Team)*

- Play → team plan based on game state
- Roles → ‘attacker in possession’
- Tasks → ‘advance the game’

***Decision Making**: how to fulfil the task*

- Skill → ‘shoot’, ‘pass’ or ‘dribble’

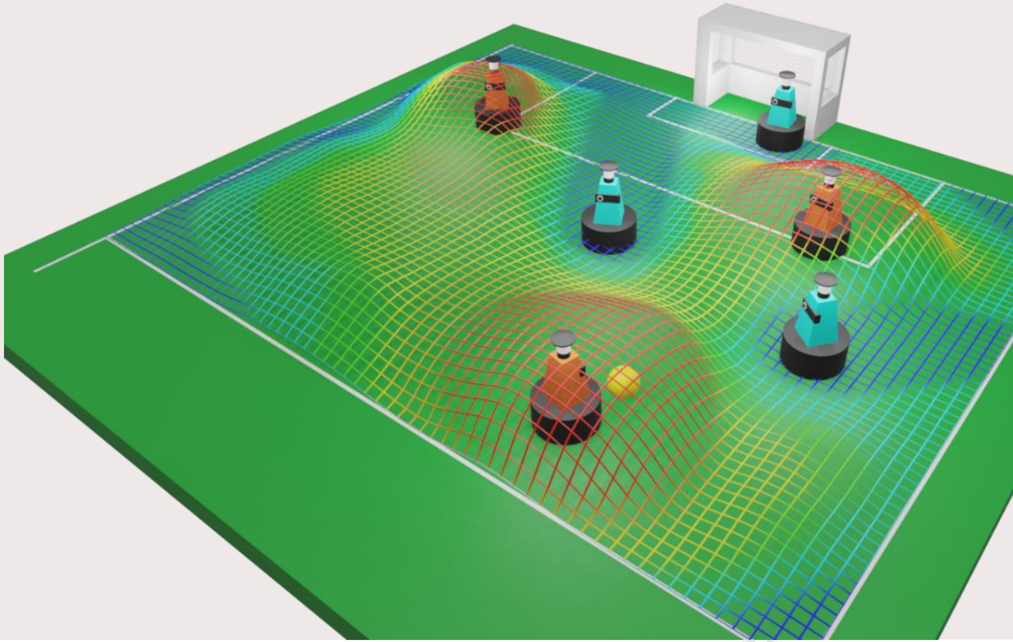
Koning, L. D., Mendoza, J. P., Veloso, M., & Molengraft, R. V. D. (2017, July). Skills, tactics and plays for distributed multi-robot control in adversarial environments. In Robot World Cup (pp. 277-289). Springer, Cham.

Mu-Fields

- For every skill a separate field
- Multiple weighted objectives
- Each position has a cost
  (red low cost, blue high cost)

$$U = \sum_{i=1}^{k} w_i * f_i(x, y)$$

- Configure strategy
  → Varying weights



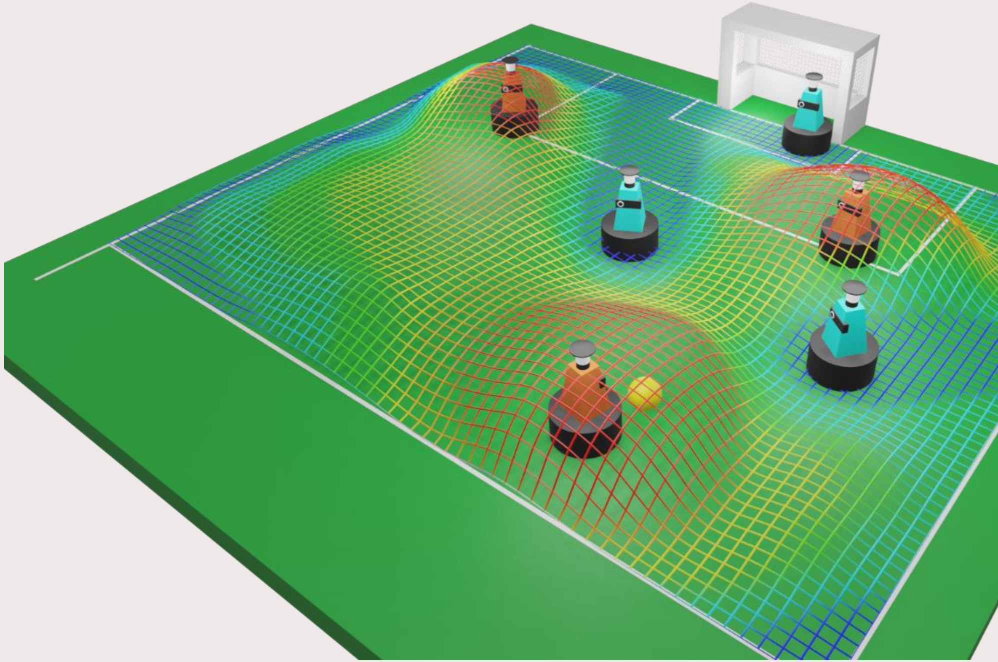
Mu-Fields

Number	Objectives	
1	Not in opponent goal area	Constraint
2	Not on opponents	Constraint
3	Not on sidelines	Constraint
4	Free line to goal	Constraint
5	Drive towards goal	Objective
6	Distance to goal	Objective
7	Free line to peers	Constraint
8	Not towards opponent	Constraint
9	Driving cost	Objective

Problem definition

- No distinct between **constraints** and **objectives**

- Redundant objectives
  → **Decreased configurability**

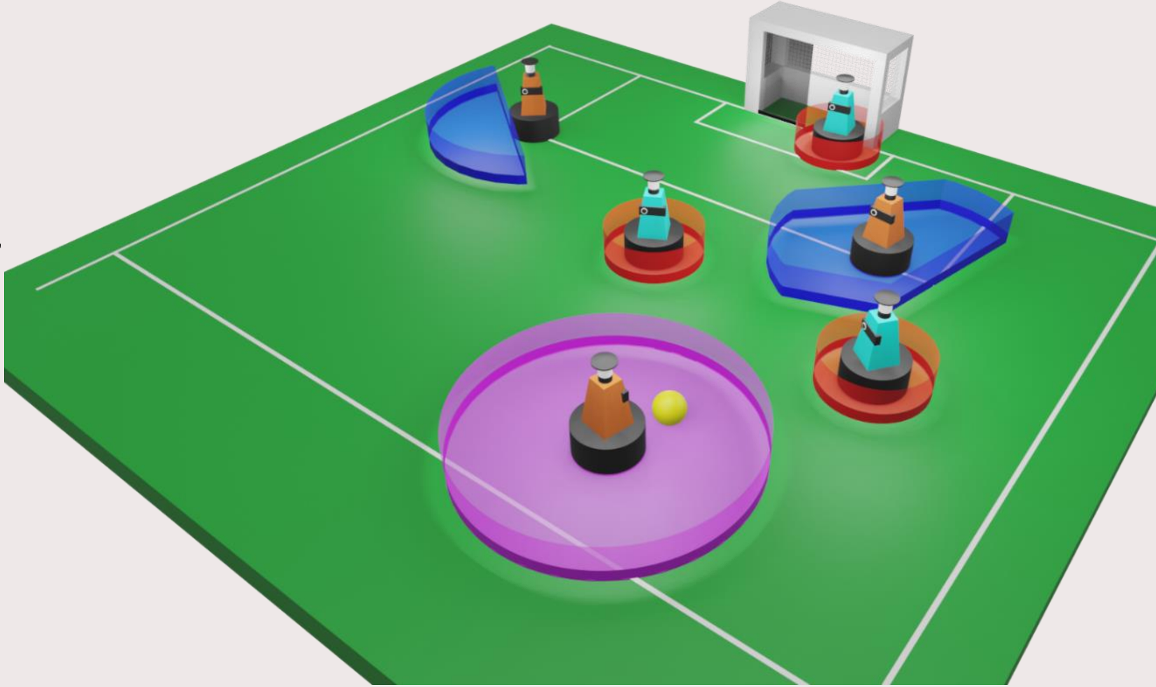


Semantic representation of skill feasibility

"Could the **strategy** of robotic football benefit from **semantic knowledge** in the **world model**?"

2D semantic regions → Feasible space

- Confined feasible space
- Less objectives



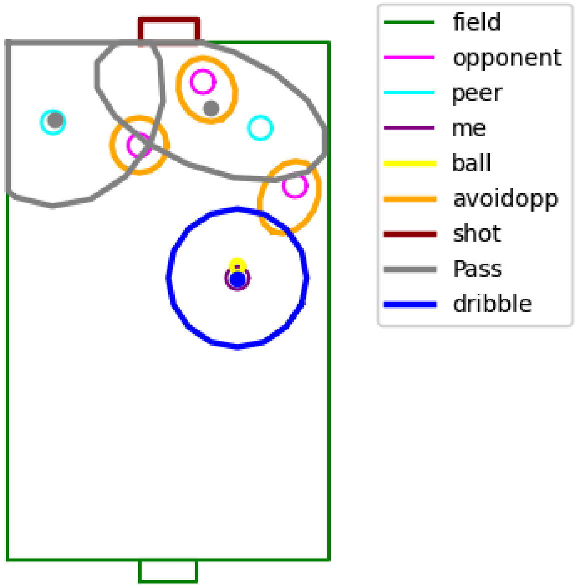
Red = avoid, black = shot, blue = pass, purple = dribble

Building the semantic map for attacking robotic football

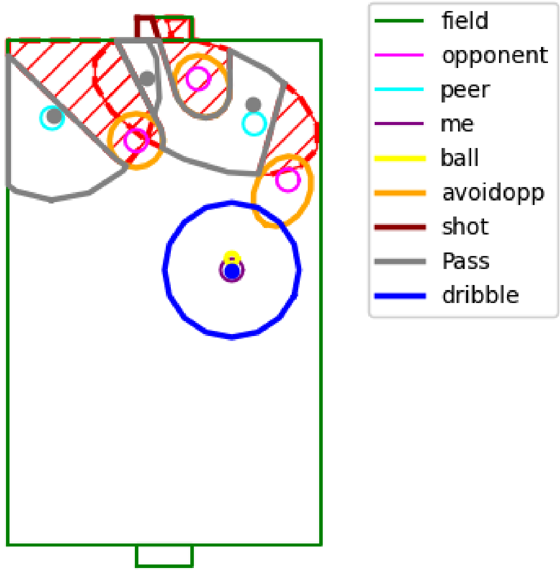
- Initial skill region
- Subtracting constraint regions

→ **Feasible space**

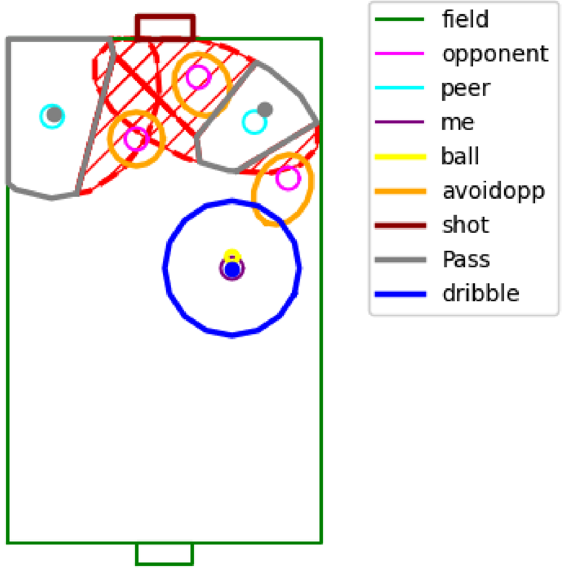
Building the semantic map for attacking robotic football



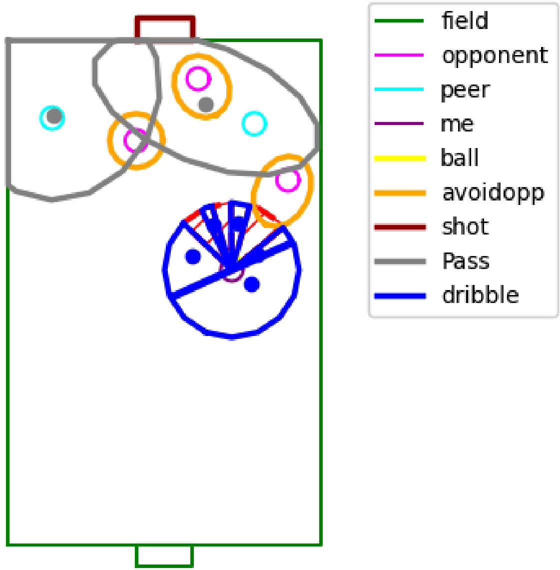
Regions behind opponents



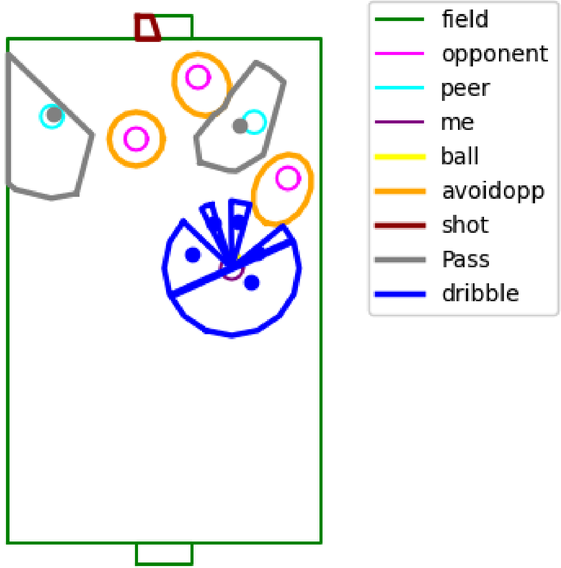
Pass regions closer to an opponent than peer

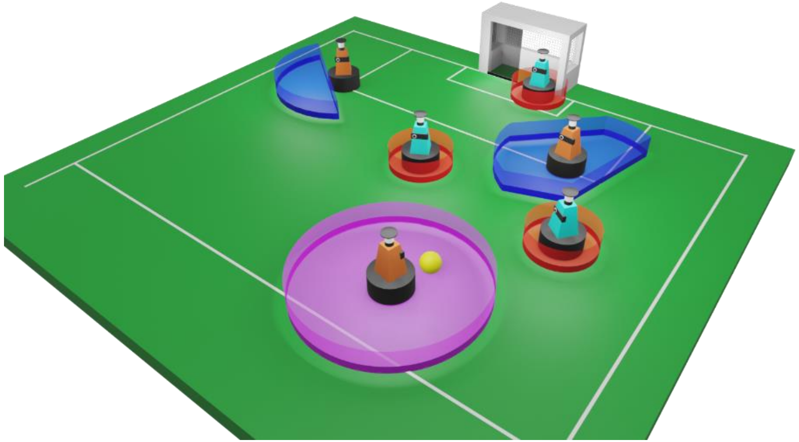


Dribble regions directed towards an opponent



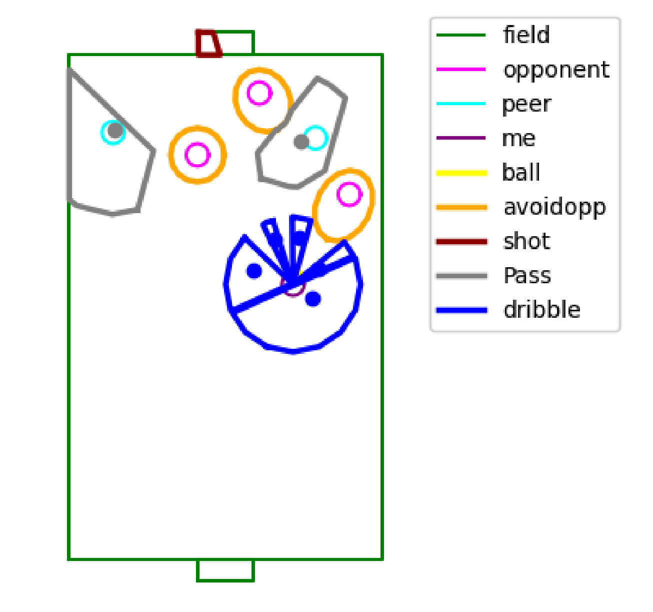
Resulting feasible space





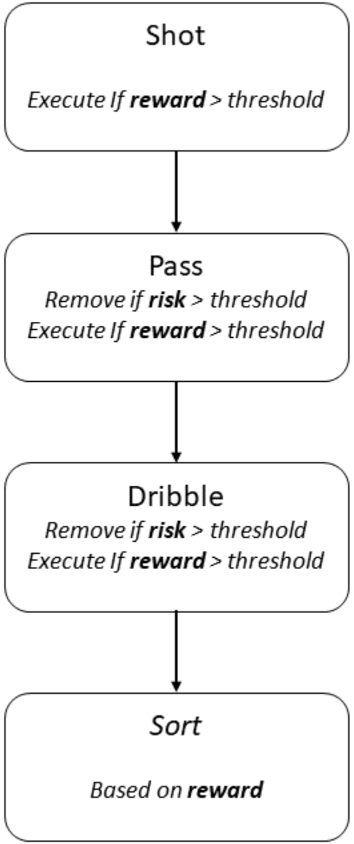
Evaluation points

- Small feasible spaces
  → **Only one evaluation point sufficient**
- Not guaranteed for other applications



Decision algorithm

- State machine
- Risk reward parameters:
  - Opponents to goal
  - Distance to goal
  - Size of the region (pressure on the ball)
  - Length of skill

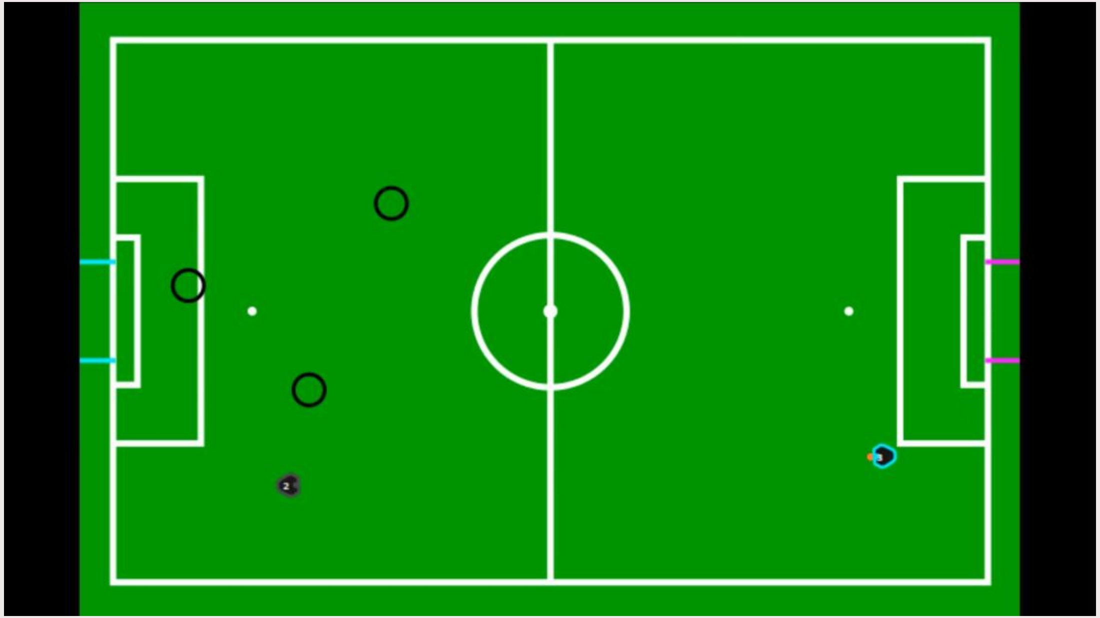


F. Goes, E. Schwarz, M. Elferink-Gemser, K. Lemmink, and M. Brink, "A risk-reward assessment of passing decisions: comparison between positional roles using tracking data from professional men's soccer," Science and Medicine in Football, pp. 1–9, 2021.

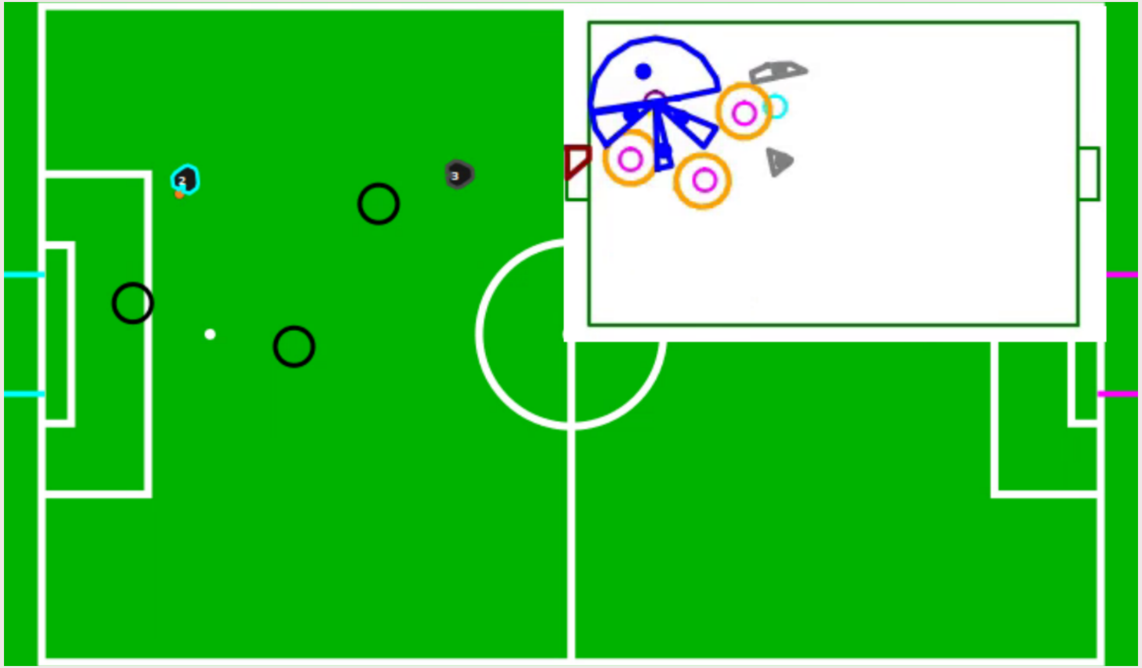
Implementation of the method

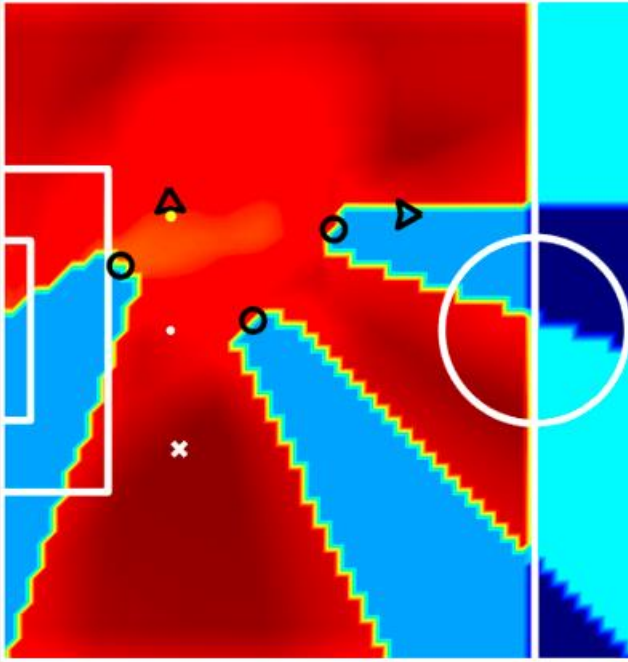
- Shapely
- Turtle simulator
- C/Python API

Results



Results





Results

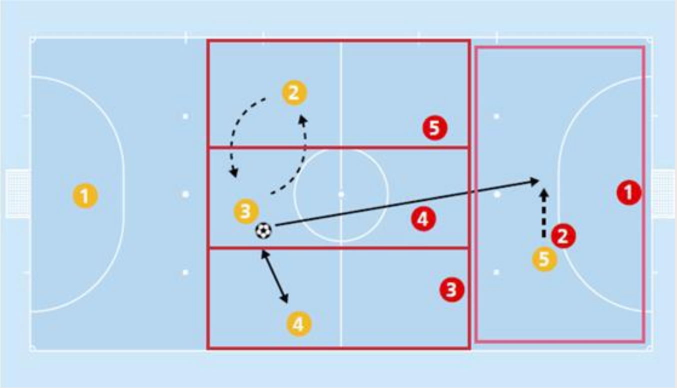
Number	Evaluation points	Constraint/ objective
Mu-fields	4978	14
Feasible spaces	1160 (8)	4

Conclusion

- Method creating feasible design space of skills in the world model
- Allows the distinction between constraints and objectives
- Less objectives and evaluation points
- **Enhances configurability → less tuning**

Future work

- Decision algorithm
- Players off the ball
  → Formations
- Hierarchical constraints (managing exploding constraints)
- Learning techniques/ knowledge based



Source: FIFA Futsal coaching manual

L. P. Reis, N. Lau, and E. C. Oliveira, “Situation based strategic positioning for coordinating a team of homogeneous agents,” in Balancing Reactivity and Social Deliberation in Multi-Agent Systems, (Berlin, Heidelberg), pp. 175–197, Springer Berlin Heidelberg, 2001.

Thank you for your attention!

For more information: **j.p.v.d.stoel@student.tue.nl**

@TechUnited

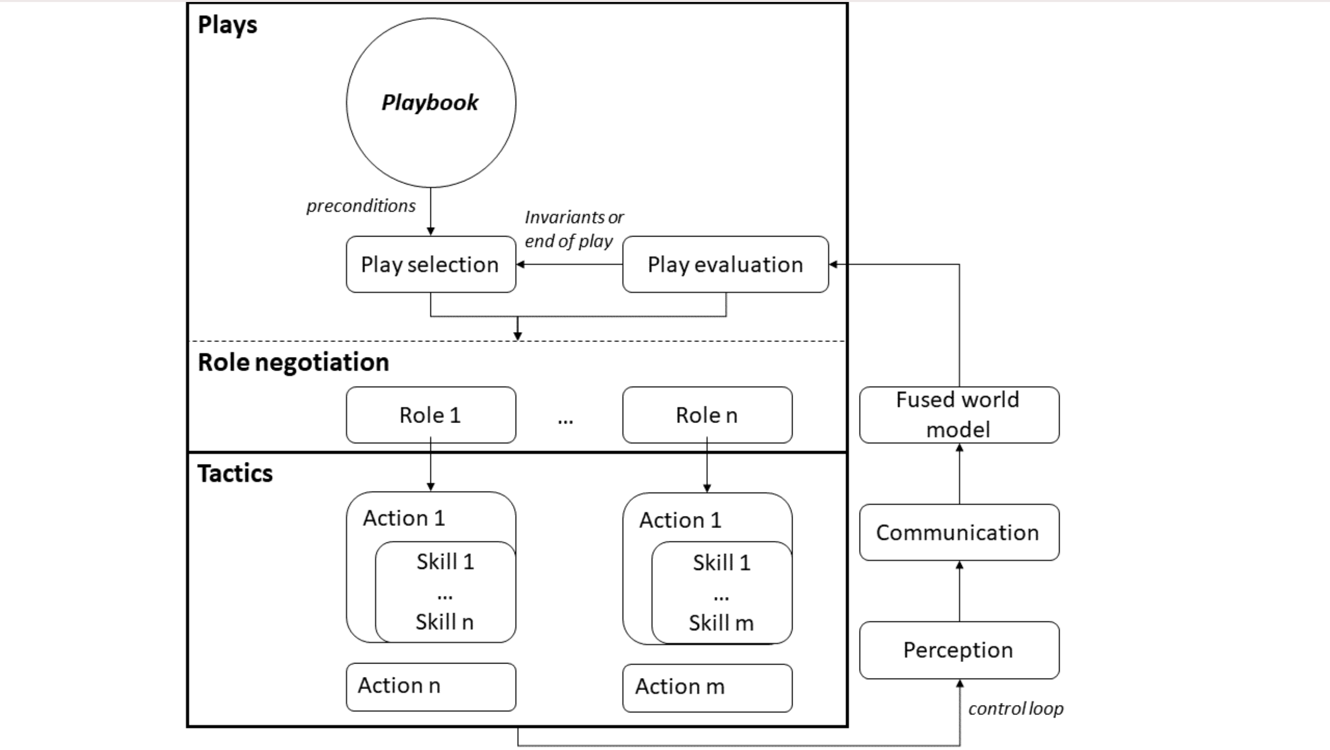
techunited@tue.nl

www.techunited.nl

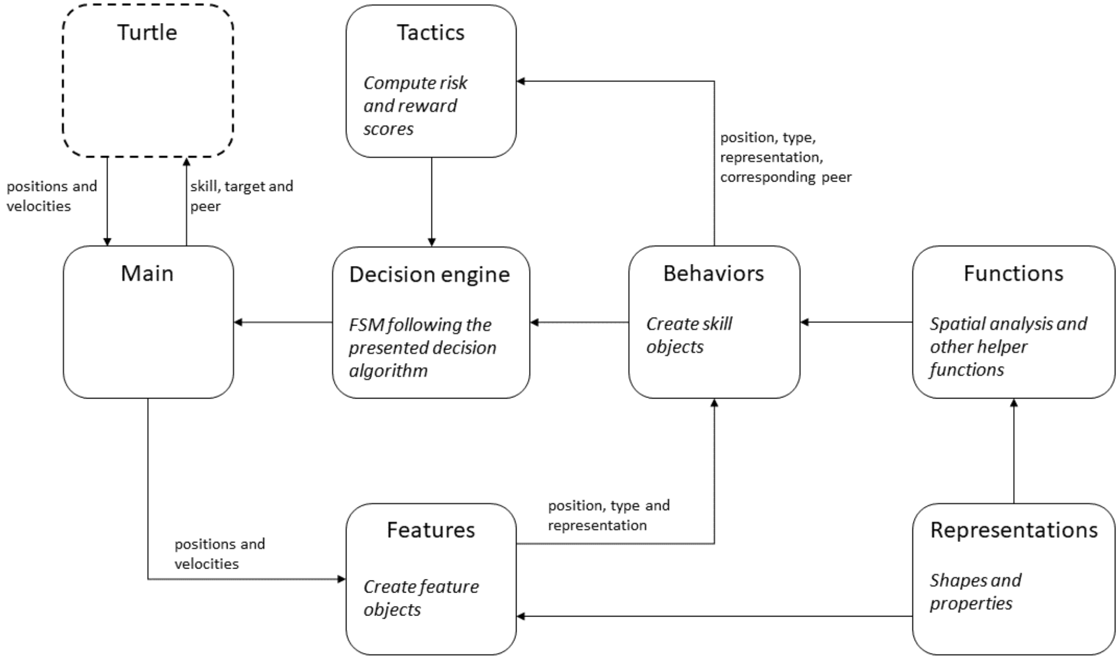
Tech United Eindhoven

Technische Universiteit
Eindhoven
University of Technology

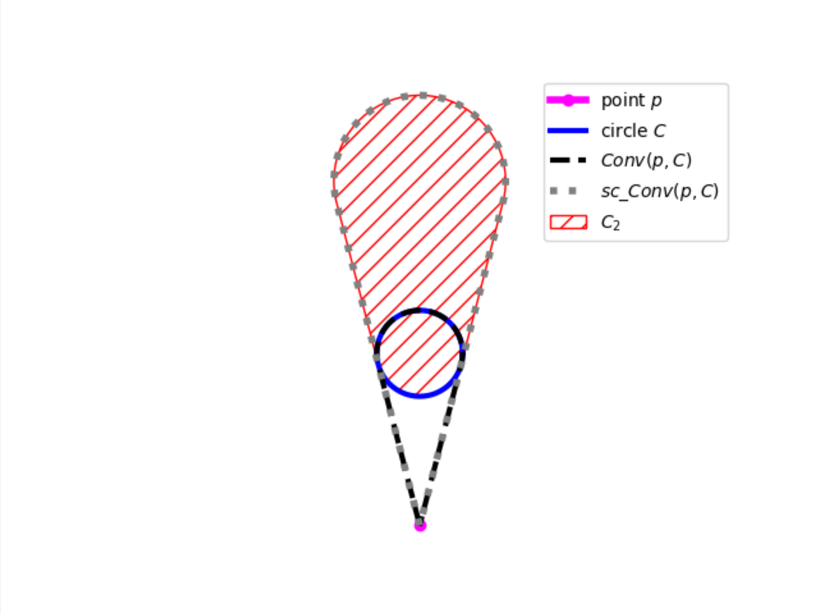
Skills Tactics and Plays



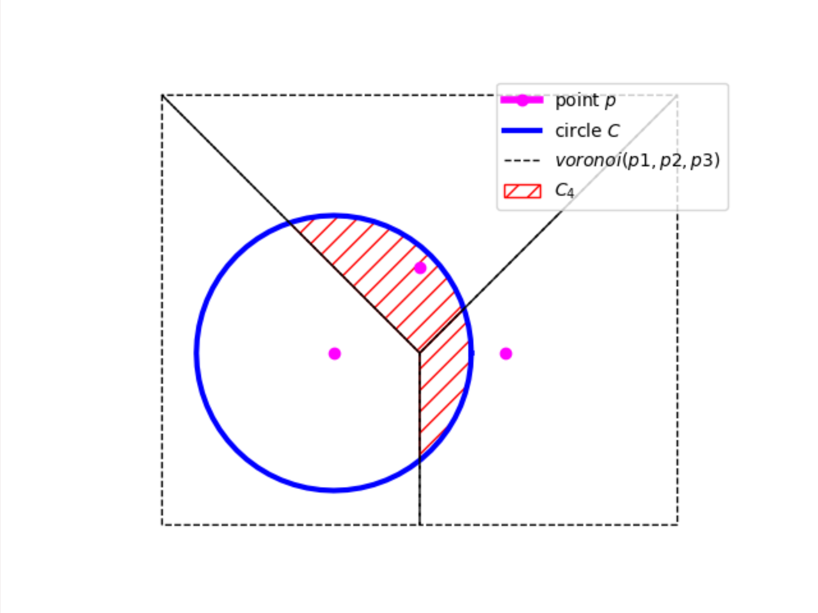
Decision algorithm



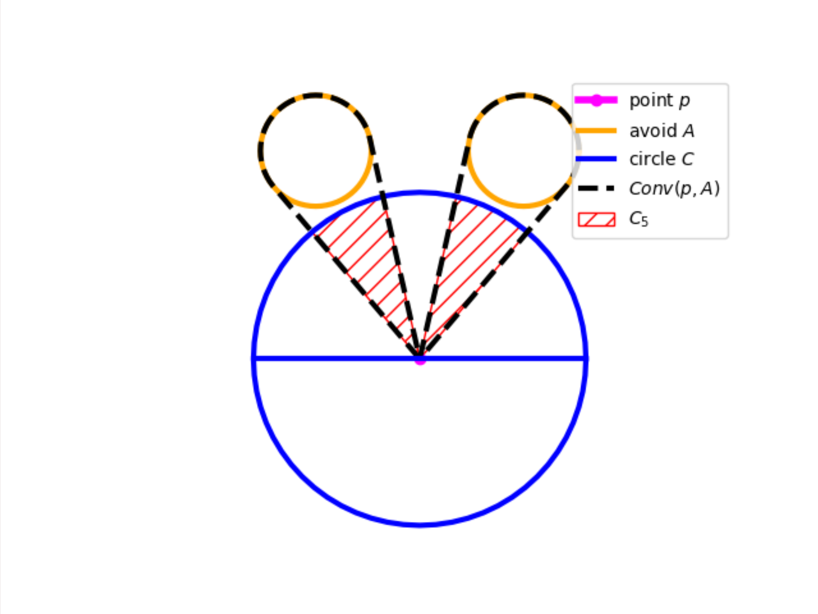
Regions behind opponent



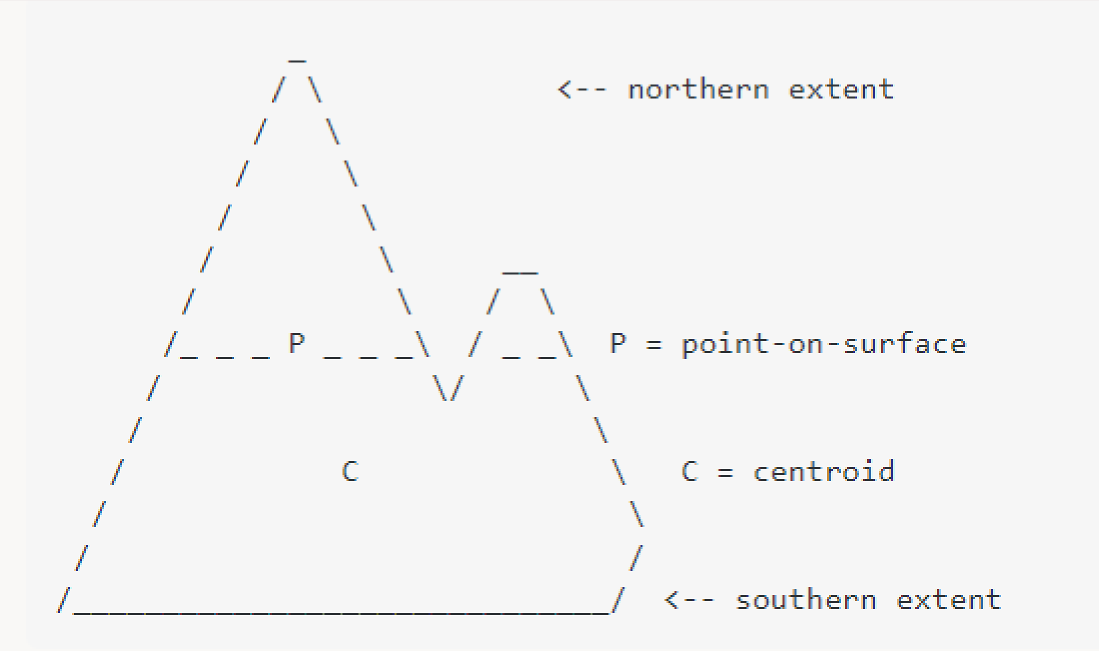
Regions closer to opponent



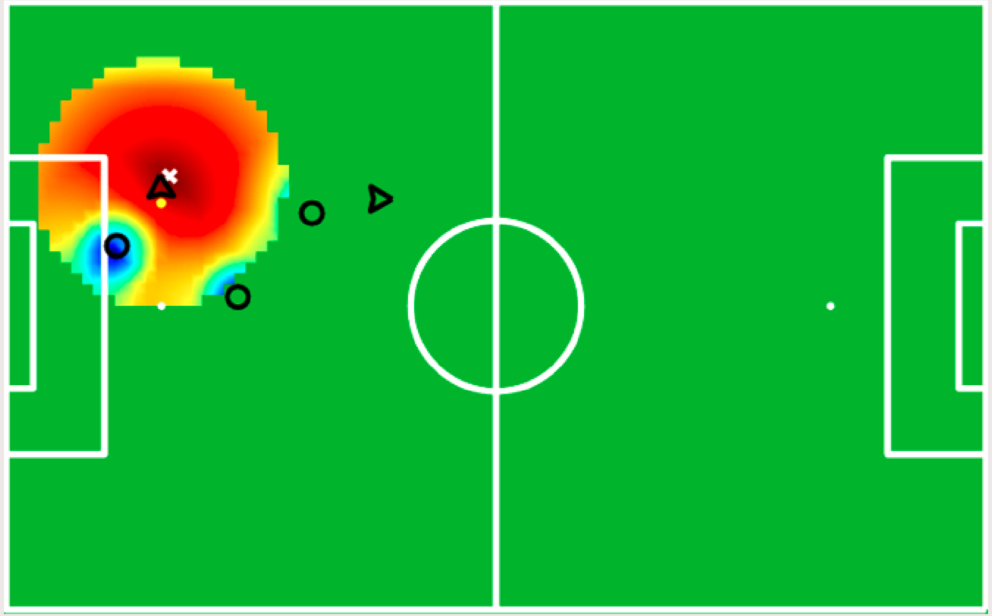
Regions directed towards opponent



Representative point function shapely



Mu field for dribbling of the snapshot from results



Snapshot code – Initializing python in Turtle software

```c
static void Advanced_Attack(pmuFieldRequest pmuRequest, pPE_InputStruct_t pS_in, pPE_OutputStruct_t pS_out, pPE_sfun_global_data_t psfgd, pSTP_TaskAdvancedAttackStruct_t pPS)
{
    clock_t time_begin = clock();
    psfgd->PyImportFlag;
    psfgd->pmodule;
    psfgd->pmeth;

    if(psfgd->PyImportFlag == 1){
    //Seth path and import python module
        const wchar_t * pName = L"Main";
        Py_SetProgramName(pName);
        Py_Initialize();
        PyRun_SimpleString("import sys; sys.path.append('/home/robocup/Turtle3/Strategy/src/SMDM/'); sys.path.append('/home/robocup/.local/lib/python3.8/site-packages/')");
        PyRun_SimpleString("print(sys.path)");
        PyObject *pmName = PyUnicode_FromString("Main");
        psfgd->pmodule = PyImport_Import(pmName);
        psfgd->pmeth = PyObject_GetAttrString(psfgd->pmodule, "run");
        if(psfgd->pmodule != NULL){
            printf("module loaded succesfully \n");
        }
        else{
            PyErr_Print();
        }
        psfgd->PyImportFlag = 0;
        Py_XDECREF(pmName);
    }

    PyObject* ppmeth = PyObject_GetAttrString(psfgd->pmodule, "run");
    /* DEBUG: Check if callable*/
    int methCheck = PyCallable_Check(psfgd->pmeth);
    if(methCheck == 0){
        printf("Methode call check: %d \n", methCheck);
    }
    /*input dictionary for Semantic Map Decision Making module (Python: Shapely)*/
    PyObject *d, *emptytup, *x, *y, *dx, *dy, *npeer, *tup0, *tup1, *tup2, *tup3;

    d = PyDict_New();
    emptytup = PyTuple_New(0);
```

Snapshot code – Initializing peer inputs for python method

```c
/* Teammates' pose and velocity */
tup2 = PyTuple_New(0);
int n = 0;
for(int i = 0; i < MAX_ACTIVE_TURTLES - 1; i++){
    if(pS_in->pTB->isActivePeer[i] && !(pS_in->pTRCB->turtleID == i + 1)){
        _PyTuple_Resize(&tup2, (n+1)*5);
        x = Py_BuildValue("d", pS_in->pTB->current_xyo[i].x);
        y = Py_BuildValue("d", pS_in->pTB->current_xyo[i].y);
        dx = Py_BuildValue("d", pS_in->pTB->current_xyo_dot[i].x);
        dy = Py_BuildValue("d", pS_in->pTB->current_xyo_dot[i].y);
        npeer = Py_BuildValue("i", i+1);
        PyTuple_SetItem(tup2, n*5,   x);
        PyTuple_SetItem(tup2, n*5+1, y);
        PyTuple_SetItem(tup2, n*5+2, dx);
        PyTuple_SetItem(tup2, n*5+3, dy);
        PyTuple_SetItem(tup2, n*5+4, npeer);
        n++;
    }
}
PyDict_SetItemString(d, "peers", tup2);
Py_XDECREF(tup2);

/* Pass in progress*/
PassInProgress(pS_in->pTB);
```

Snapshot code – Calling python method

```
reprint(d); /*print feature tuple*/
PyObject* SMDM = PyObject_Call(ppmeth, emptytup, d);
if(SMDM == NULL){
    PyErr_Print();
    printf("ERROR: Calling Python module did not succeed \n");
    exit(1);
}
reprint(SMDM);
PyObject* s  = PyTuple_GET_ITEM(SMDM, 0);
PyObject* tx  = PyTuple_GET_ITEM(SMDM, 1);
PyObject* ty  = PyTuple_GET_ITEM(SMDM, 2);
PyObject* p  = PyTuple_GET_ITEM(SMDM, 3);
int skill = (int) PyLong_AsLong(s);
double target_x = PyFloat_AsDouble(tx);
double target_y = PyFloat_AsDouble(ty);
int peerID = (int) PyLong_AsLong(p);
Pos_t actionTarget = {.arr = { target_x, target_y}};
pPos_t at = &actionTarget;
printf("skill = %d, target x = %f, targe y = %f, peer = %d \n", skill, target_x, target_y, peerID);


Py_XDECREF(emptytup);
//Py_XDECREF(d);
Py_XDECREF(SMDM);
Py_XDECREF(ppmeth);
```

Snapshot code – State machine

```
def onBall(self, ball, peers, opponents, field):

    remove = []
    Behaviors.avoidOpponent(opponents)
    Behaviors.avoidPeer(peers)

    Behaviors.makeShot(self, field)
    shotList = Behaviors.shot.get()
    for shot in shotList:
        shot.reward = Tactic.scoreReward(ball)
        print(f"considered a {shot.type} to {shot.pos} from {ball.pos} with score reward {shot.reward}")
        if shot.reward > shotReward_treshold:
            return shot

    Behaviors.givePass(self, peers)
    givePassList = Behaviors.Pass.get()
    for givePass in givePassList:
        givePass.reward = Tactic.scoreReward(givePass)
        givePass.risk = Tactic.totalRisk(self, givePass)
        print(f"considered a {givePass.type} to {givePass.pos} with score reward {givePass.reward} and total risk {givePass.risk}"
        if givePass.risk > passRisk_treshold:
            remove.append(givePass)
            continue
        if givePass.reward > passReward_treshold:
            return givePass

    Behaviors.makeDribble(self)
    dribbleList = Behaviors.dribble.get()
    for dribble in dribbleList:
        dribble.reward = Tactic.scoreReward(dribble)
        dribble.risk = Tactic.totalRisk(self, dribble)
        print(f"considered a {dribble.type} to {dribble.pos} with score reward {dribble.reward} and total risk {dribble.risk}")
        if dribble.risk > dribbleRisk_treshold:
            remove.append(dribble)
            continue
        if dribble.reward > dribbleReward_treshold:
            return dribble

    skills = [*shotList, *givePassList, *dribbleList]
    for skill in remove:
        skills.remove(skill)

    return CompareSkills(skills)
```

Snapshot code – Update features

```python
def update(inputstruct):
    ballpos, ballvel, mepos, mevel, peerpos, peervel, peerlab, opponentpos, opponentvel = Feature.convertInputs(inputstruct)

    ball.pos = ballpos
    ball.vel = ballvel

    me.pos = mepos
    me.vel = mevel

    for i, peer in enumerate(peers):
        try:
            peer.pos = peerpos[i]
            peer.vel = peervel[i]
            peer.lab = peerlab[i]
        except IndexError:
            Feature.delete(peer, peers)
        except:
            print("Not able to update peer")

    nNewpeers = len(peerlab) - len(peers)
    if nNewpeers > 0:
        for i in range(nNewpeers):
            Peer(peerpos[-i-1], peervel[-i-1] , peerlab[-i-1])

    for i, opponent in enumerate(opponents):
        try:
            opponent.pos = opponentpos[i]
            opponent.vel = opponentvel[i]
        except IndexError:
            Feature.delete(opponent, opponents)
        except:
            print("Not able to update opponent")

    nNewOpp = len(opponentpos) - len(opponents)
    if nNewOpp > 0:
        for i in range(nNewOpp):
            Opponent(opponentpos[-i-1], opponentvel[-i-1])
```

Snapshot code – Behavior object

```
#SETTINGS
Woppvel = 0.8
avoidOppDiameter = 1.2
avoidOppForShotDiameter = 0.5
avoidPeerDiameter = 0.8
fv = 0.5

DRIBBLEDIAM = 3
MINPASSDIST= 3

DRIBBLEREGION = None
INFLUENCEDIST = 15.0

class Behavior:
  all = {}

  def __init__(self, shape, pos, peer = None, shape2 = None):
    self.pos = pos
    self.peer = peer
    self.type = self.__class__.__name__
    self.ntype = -1
    self.shape = shape
    self.opponentsToGoal = 0
    self.shape2 = shape2
    if self.type in Behavior.all:
      Behavior.all[self.type].append(self)
    else:
      Behavior.all[self.type] = [self]

  def __repr__(self) -> str:
    return self.type

  def delete(self):
    Behavior.all[self.type].remove(self)
    del self

  def clear():
    Behavior.all.clear()

  @classmethod
  def get(cls):
    name = cls.__name__
    if name in Behavior.all:
      return Behavior.all[name]
    else:
      return []
```

Snapshot code – Behavior give pass

```
"""Put circle around peer to indicate locations for which a pass can be given to that peer """
def givePass(me, peers):
  time = t.time()
  peerList = peers
  for peer in peerList:
    if Shape.distance(me.pos, peer.pos) > MINPASSDIST:
      radius = fun.calculatePassRadius(me, peer, fv)
      shape = Shape.ellipse(peer.pos, radius, peer.vel, fv)
      shape = inField(shape)
      action = Pass(
        shape, None, peer
      )
      pos = fun.passPoint(action)
      action.pos = pos

    checkReachability()
    checkPassLine(me)
```

Snapshot code – Behavior check passline

```
"""
Check if no player (only opponents for now) is blocking a pass line.
Covex hull of the pass region around a peer with the ball position is taken.
If intersection the pass region is reshaped by taking the tangent lines (by use of convex hull) to the avoid region of the opponen
"""
def checkPassLine(me):
    passList = Pass.get()
    avoidList = avoidopp.get()
    for action in passList:
        availableRegions = fun.freeLine(me.pos, action.shape, avoidList)
        if availableRegions == None:
            Behavior.delete(action)
        else:
            for i, region in enumerate(availableRegions):
                if i == 0:
                    action.shape = region
                    action.pos = fun.passPoint(action)
                else:
                    action = Pass(region, None, action.peer)
                    action.pos = fun.passPoint(action)
```

Snapshot code – function calculate free line

```
def freeLine(startPos : float, actionRegion, objects : list):
    blocked = []
    startPos_tup = [tuple(startPos)]
    EX_action = actionRegion.exterior.coords[:]
    CH_points = MultiPoint(startPos_tup + EX_action)
    CH_action = CH_points.convex_hull
    for avoid in objects:
        avoidRegion = avoid.shape
        if CH_action.intersects(avoidRegion):
            EX_avoid = avoidRegion.exterior.coords[:]
            CH_points = MultiPoint(startPos_tup + EX_avoid)
            CH_avoid = CH_points.convex_hull
            SC_CH_avoid = scale(CH_avoid, 10, 10, origin= startPos_tup[0])
            if SC_CH_avoid.contains(actionRegion):
                return None
            else:
                blockedRegion = Shape.buffer(SC_CH_avoid.difference(Shape.buffer(CH_avoid.difference(avoidRegion),0.01)), 0.02
                blocked.append(blockedRegion)

                #DEBUG
                if DEBUG:
                    constraint = actionRegion.intersection(blockedRegion)
                    Functions.plotconstraint(constraint, 'red')

    actionRegions = [actionRegion]
    for region in blocked:
        if actionRegions == None:
            break
        actionRegions = Shape.differenceMultiPolygon(actionRegions, region)
    return actionRegions
```