# Notes

**Code:**

* From time to time, it seems as if the collision between target and agent isn’t detected, yet that can be due to the low number of targets, implying that the new target appears on the same place.  
  However, one observation is that when it occurs, it seems as if the target needs to be collided with three times, for the collision to be acknowledge, and it seems implausible that the target appears the same at place three times in a row.
* The reason for not resetting the position of the target when the game is being reset, is because chosen not to do so forces the agent to learn difficult part of the environment. It is postulated, by myself, that if the position of the target was being reset every time the agent collide with the environment, the agent could only learn the easy parts of the environment – because the agent, most of the time, would collide with the environment on the difficult parts of the environment.

**Game:**

* When the target is in narrow passages, i.e. between the upper/lower horizontal wall and the first/last obstacle, it gets hard for the agent to locate the target, which could be a motivation for curriculum learning. The idea could be to learn part of the environment first, then add difficulty and thereby improve learning.

**Challenges:**

* ***What behaviour should the pedestrians exhibit, i.e. how should they move?***

Current status: The agents are NavMesh agents, and they move to random locations within the configured NavMesh every predefined timestep – predefined by the researcher.

* ***Transition between brains – how?***

Right now, it is a heuristic transition.

* ***Parallelizing/Gradient-Based-training;*** A trade-off between important information getting lost in a feedback loop or in irrelevant experience.

**Assumption:** Curriculum learning is used

***What can be done?***

Parallelization of the training possesses several benefits, such as accumulation of more diversified experience. Yet, the effectiveness is controlled by two parameters, the threshold for changing lessons and the maximum number of steps allowed for the agent to take, trying to locate the goal. Choosing far from optimal values can result in sub-optimal policy functions.  
But how should these two parameters be chosen? They could be chosen arbitrary, hoping for a lucky strike, or we could see if we could generate data to indicate ranges for the two parameters.

Running standard single environment training sessions, generating data of number of steps used reaching the goal, would give an idea about with what range within the two parameters should be chosen.

Example: Say a single environment training session has been conducted, and it reveals that, on average, it takes 1000 steps reaching the goal. With that in mind, and the distribution of step taken, that gives an idea about within what range the maximum number of steps should be chosen. Furthermore, every step is penalised with a time penalty of 0.0005, implying that the average reward should converge to 0.5 (1000\*0.0005), which tells us that a threshold near 0.5 or above seems unreachable.

The above analyses could be done for different environments with varying complexity, and for multiple runs, to enable a more rightful and stable decision.

* ***Hierarchical RL agent - Design of the reward structure of the low-level agent.***

One way to do it is to let an *invisible agent* move as suggested by the high-level agent, and then keep track of the position of this agent and our actual agent. The reward to the low-level agent should be proportional to the differences, in degrees, between the suggested path and realised path.

However, using an invisible agent can potentially introduce bias in the reward structure, because the actions suggested by the high-level agent is based on its own position, and not that of the invisible agent.

*The reward to the low-level agent should be generated by the high-level agent and should reflect whether or not the low-level agent achieved travel in the desired direction, and if so, the speed with which the low-level agent achieved the goal.* (Yen and Hickey, 2004).

Right now, is the penalty determined by the high-level agent but invoked by the invisible agent. But speed is not rewarded.

***One question for now:*** Should the environment reset if the invisible agent collides with walls/obstacles?

***Another question:*** Should the low-level agent receive more frequent feedback than the high-level agent, or should the reward signal be just as sparse? (Bonus: Interesting to see what the effect of that is).

* ***How to model uncertainty.***

From (Garcia and Fernández, 2015) a way to model the uncertainty is using a modification of the traditional optimization criterion named *Worst case Criterion* (WCC). WCC can capture the variability caused by parameter uncertainty related to some of the parameters of the MDP, which could be useful in this case here. It would be useful if the distance to nearby sensors was always a part of the observation at time t.

Another way, also highlighted by (Garcia and Fernández, 2015) could be to use *Constrained Criterion* (CC), in which the optimization occurs while keeping some other parameter/-s above/beyond some defined bound. For the case of interest, that could be to keep a certain distance to sensor centre’s if the expected reward of approaching is less than the intrinsic reward.

* ***More complex environments, both in terms of size and the degree of how dynamic the environment is.***

**One observation**, presumably occurring because of the how the environment is designed, is that when the target is located between either, a wall and the first obstacle or a wall and the last obstacle, which is a more narrow space than between any two obstacles, then the agents struggles to find the target – even after 350000 steps.

**Within this challenge, is how to settle on the maximum number of steps allowed:**

Remember the awful training in FTS-1.2. Setting the maximum number of steps to 1000 seems to solve the issue, and we can move on – but no. If 1000 is the magic number, we would expect to see the length of completed episode decrease as learning to place, and looking at the data generated from FTS-1.3.4 or 1.4.3, that is not the case.

* ***Choosing parameters; Target Frame Rate (Academy), Time Scale (Academy), Decision Interval (Agent) and Max Steps (Agent).***

***Target Frame Rate:***

*Target Frame Rate is the frame rate that the Unity engine will attempt to maintain. This means that the engine will make a best-effort attempt to run all game logic and physics updates and also render within the time of 1 / target\_fps seconds. If this value is set too high, there is a possibility you could see unexpected physics simulation behavior.* Source: <https://github.com/Unity-Technologies/ml-agents/issues/1425>

**General:**

<https://docs.unity3d.com/ScriptReference/Application-targetFrameRate.html>

***Time Scale:***

***Decision Interval:***

***Max Steps:***

**To-do:**

* ~~Include the distance to the sensor centre in the observations received, and thereby do the distance calculation.~~ 
  + **~~Add graph to TensorBoard~~** ~~showing the frequency of the agent colliding with the sensors, and how often the target is located inside the sensors. (We want to see that the attraction to the target is larger than the aversion to the sensors).~~

**Is for now implemented in pure Python instead.**  
**One behaviour that is observed is:** The agent seems to get stuck when the target is located *behind* the sensors.

* **~~Adjust the trail~~** ~~to not include the jumps back to the start position.~~
* **~~Allowing more sensor clouds to appear (Started)~~**
  + **~~It does not work at the moment because I tried to add individual tags for each cloud, to easier change the position of each cloud, and not just of all sensors.~~**

**~~It is however a bit challenging to add tags at runtime, which is the struggle right now.~~**

* **~~Look into colouring the trail based on the frequency of trails in a certain area.~~**

Implemented as adjusting the alpha of the colour trail, in batches equally divisible with 10.

* **~~Add Visual input to the agent~~** (Way to slow, in terms of training time).
* ~~Add dynamic obstacles (Pedestrians)~~
* Look into how challenging it is to implement new RL methods in TensorFlow.
* ~~Change position of target completely randomly, and not using targetSpawns.~~
* ~~Create a visual appealing infographic of your ideas, to be used when the idea is to be present for Ed.~~ (Didn’t happen before the meeting and is so disregarded).
* Look into how to create 3D cities in Unity (best practice).
* Look into the collision between the agent and the pedestrians – it seems as there is some sort of contraction at the point of collision. **(Hasn’t been a problem for some time)**
* ~~Curriculum learning – can it benefit the learning process here?~~ **(Indeed it could)**
* ~~New Game objects are mysteriously being created doing runtime.~~
* ~~Shrink the x and z extend a bit for the NavMesh agents, as they currently are allowed to go all the way out of the environment.~~
* Fix the memory expensive feature of the custom trail-drawing implementation.
* ~~Implement such that the reward from colliding with the sensor clouds depends on the number of sensors in the cloud. <- pre-step to implementing the uncertainty aspect.~~

~~See line 67-68 in the Academy script for an idea.~~

Implemented; but if it is perfect is another question.

* Implement two brains, for the hierarchical RL implementation.

~~One idea; Use GiveBrain() to change between brains. Train each brain using two different curriculums.~~

Implemented; With heuristic transition between brains.

* ~~Consider resetting the position of the target on reset and add a number for maximum steps allowed for the agent. (Prepared for – just test it – Not working now – the new position is off).~~
* ~~Fix that only one sensor cloud appears, and not one in every area.~~
* Implement two agents in one GameObject, as an alternative hierarchical RL implementation.
* ~~Train with concurrent training sessions:~~ <https://blogs.unity3d.com/2019/04/15/unity-ml-agents-toolkit-v0-8-faster-training-on-real-games/>

Train with multiple training areas within one application instead.

* Generate collision statistics for the three cases of curriculum learning.
* ~~Check if OnCollisionExit invokes after collision with sensors.~~
* Investigate the effect of add additional training areas as part of the curriculum, to minimize the feedback-loop effect.
* ~~Investigate how invoking max steps effects the full training session.~~ Good effect of even too high a bound – Interesting!
* Investigate the effect of bounding the penalty (The bad training in FTS-1.2 could be due to over-penalisation)
* Investigate if the size of the observation vector can be increased doing runtime, and if it can, incorporate information (presumably distance) about sensor clouds.
* Investigate the effect of using different degrees of density in the sensor clouds instead of varying number of sensors. **This is ready for test in the morning – could end up making the task two above redundant. Right now, one density is present but more can easily be added and it is worth to think about how different densities should be incorporated (randomly, with a fix share of each maintained after reset etc.)**
* ~~Re-run FullTrainingSession1.2 (unbounded) and 1.3 (ms: 1000) again and then run FullTrainingSession-1.4 for the first time.~~ **SensorCollision stats are inaccurate because no recording was done when colliding with the actual sensor..**
* ~~Re-run FTS-1.3 with continuous collision detection, under the tag 1.3.2.~~

~~Didn’t help – now trying to lower timescale to 50, under the tag 1.3.3.~~

~~No luck – Could be because collision with sensor still was under “trigger” – now moved to “Collision” and timescale at 60, under tag 1.3.4~~

The conclusion on the fact that multiple collisions are recorded, even though no more than one should be possible, is that it happens because the area is so dense as it is. Having two sensors placed right next to each other is plausible. If this is to be eliminated completely, a more complex placing of the sensors is needed, where the relative position of the sensors are taken in to account.

* ~~Test FTS-1.4 with a speed of 5 in crowded areas instead of 2.5 as initially.~~

1.4.1: speed 5; Nothing

1.4.2: Completely remove initialisation of sensors from code; no change – good

1.4.3: No change in speed, only increased penalisation – less noisy training – good, yet still two significant dropouts.

1.4.4: Monitored training process in the editor. **Cancelled**

1.4.4: Monitored training process in the editor. Added different materials for each of the three possible densities and upped the number of parallel training environments to 4 at that point.

* ~~Run CLStaticDynamic with mass of pedestrians at 70 kg and drag at 2.5.~~
* It could be an idea to fix the location of the crowded area and see if that helped on learning to avoid.
* ~~Include the idea of “digital twin” in the Unity section.~~
* Test visual observations again
* ~~Make sure if PPO is on-line or off-line~~
* ~~Include a section in appendix on choosing batch/buffer size and related parameters.~~
* Test ideas for aiding learning~~:~~ **~~Increasing number of areas using the curriculum~~**and adding an additional action in agent (direction of movement), use visual observations as well, reward shaping (RS) as of (Mirowski et al., 2018)[[1]](#footnote-1), ~~vector stacking~~.  
  **RS not first priority.  
  ~~First vector stacking is running – but should the buffer size be increased proportionally to number of stacked vectors?~~ No**
* ~~Run one more run of full-parallelised set-up with four times larger buffer size~~
* ~~Depending on the outcome of above, run one more of CcCLStaticDynamic-1.5~~
* Why RL? Partially described in the introduction.
* REMOVE THE PART ABOUT FEEDBACK-LOOP-EFFECT IF SPACE BECOMES A CONCERN, AS IT REALLY SHOULDN’T BE A PROBLEM IN THIS APPLICATION, NOT AFTER ALL THE INTIATIVES PRESENTED TO AID THE TRAINING PROCESS.
* Finish parallelisation figure.

**Model setup:**

**Idea 1:**

Hierarchical setup: two different levels, low-level and high-level.

High-level takes care of the overall navigation and the low-level takes care of the pedestrian avoidance. Each level would potentially employ two different versions of RL, and the decision to take actions, resulting in the need to use the low-level part, should be based on a comparison between the expected reward from previously interactions using the low-level part and the intrinsic reward (from curiosity learning).

**Idea 2:**

Standard PPO setup as now, with the distance to the sensor centre included in the observation at time t. If the distance is less than some defined threshold, the agent receives a modest negative reward and if the agent collides with the sensors, a greater negative reward is received.

This way simulates the hierarchical setup, where it is assumed that it is more costly to safely navigate around all pedestrians in a crowded area.

The way the uncertainty can be incorporated is to let the end point of the distance calculation be chosen stochastically, i.e. if the probability is one, the centre is chosen, else, any point is chosen with equal probability. That would imply uncertainty about where the low-level action is needed (from the idea above), which is here reflected by the risk of receiving a negative reward upon unexpected collision.

**Focus:** *Italic notes symbolise the take-away’s from the chat with Ed.*

* PPO
  + *For now, don’t aim at implementing new extensions of RL.*
  + *~~Focus on using curriculum learning~~ and added an extra brain, to try out the global/local navigation (sort of hierarchical learning).*
* Accessing the opportunities within Unity for RL, by building complex environment/-s.
  + With dynamic obstacles
  + *~~Include dynamic obstacles in the toy environment but do not focus on building a full-size city for the learning to take place in.~~*
* Uncertainty in the location of crowded areas.
  + *Negative reward depending on the number of people.*
  + *If the above was the case, uncertainty could be included by added noise to the information the agent receive, in terms of the density of the sensor clouds around it.*
* Accessing the power of PPO

**Implementations:**

**Base:**

* No sensors
* Target
* Obstacles
* Walls
* Observations as vector input

**SensorEnvironment:**

* Target
* Obstacles
* Walls
* Observations as vector input
* Dynamically added sensors in each state

**SensorEnvironment-2.0:**

* Target
  + Randomly changes position instead of using predetermined spawning places.

For now, the target is not allowed to be located within an obstacle or wall, which could be allowed to represent that the target is located inside a building.

* Obstacles
* Walls
* Observations as vector input
* Dynamically added sensors in each state
* Dynamic obstacles, in form of *pedestrians*.
* Visual input added to the observations received at each state (Not tested yet). (Camera removed for now)

**SensorEnvironment-3.0:**

In addition to all the previously listed:

**Curriculum learning**

* Adding of sensors clouds and moving sensors has been moved into the academy script, to allow the use of *resetParameters*.
* Spawning from the academy script has been generalised to apply to multiple areas.

**SensorEnvironment-4.0:**

In addition to all the previously listed:

* Negative reward of colliding with sensors depends on the number of sensors collided with.
  + Minor negative reward for every collision with sensors, twice the size of the negative reward received when exploring.
  + Negative reward for entering *CrowdedAreas*, which depends on the number of sensors collided with.
  + The noise added depends on the local interactions (sensorCollisionsLocal).

**Future work:**

* + Optimal construction of the reward received and hereof the size of the negative reward received for colliding with the individual sensors.
  + Should the noise added depend on the local interactions (sensorCollisionsLocal) or the worst-case scenario (Number of sensors per cloud).
* Added a feature to locate if the target is in the difficult region (located between either, a wall and the first obstacle or a wall and the last obstacle)
* The penalty for colliding with sensors has been increased 100 times.

**SensorEnvironment-4.1:**

* Changed the reward (penalty) system associated with colliding with the sensor clouds and increase the magnitude of the penalty.

**SensorEnvironment-4.2:**

* Moved away from sensor clouds and over to density clouds, as one way to bound the overall magnitude of the penalties. This was done in response to the lack of improvement in FTS-1.2, which could be a result of over-penalisation, despite limiting the number of steps allowed for the agent to take.
* Added individual materials for each of the current three possible densities that the areas can possess.

**SensorEnvironment-5.0:**

* This implementation is made ready for hierarchical RL, which sadly isn’t supported by Unity yet – or at least training of it.

**Unanswered questions:**

* Should the agent be reset when colliding with sensor clouds?
* Should the environment reset if the invisible agent collides with walls/obstacles?
* Should the low-level agent receive more frequent feedback than the high-level agent, or should the reward signal be just as sparse? (Bonus: Interesting to see what the effect of that is).
* How thorough should the outline of the methods used be? PPO, q-learning and TRPO
* What is your take on deviations from the traditional headers/outline of a project?  
  *Setting the Stage* as a header for the introductory section, *Prerequisites* instead of *Methodology* and Exploration/Exploitation trade-off instead of *Analysis* *(Results/Discussion)* and *Policy Evaluation* instead of *Conclusion*.
* What is your take on small transition paragraphs between sections?  
  Sort of: *This serves as good time to shift attention to Unity, which is the centre of the following section.*
* ~~Comparing the description of the academy and the agent, does it work better with specifically listing each element being described, as in the description of the agent?~~
* **A thing for future investigation:** The effect of the relative size of the target.

1. Third paragraph describes exactly how. [↑](#footnote-ref-1)