3806ICT - Exercises week 1 $\rightarrow$ 5

Nick van der Merwe - s5151332 - nick.vandermerwe@griffithuni.edu.au

April 22, 2021

Question 1

Locomotion is defined as a robot's ability to move itself by exerting force on the environment whereas manipulation is its ability to move objects by exerting force upon them. So their shared feature is that they exert force using a mechanism, but their difference is what ends up moving. In locomotion, the robot itself moves and in kinematics the other object moves.

Question 2

Easiest format to see this in would be lists:

Advantages

- They can go over more complicated obstacles without getting stuck (slanted ground, steps, et cetera)
- Causes less damage to terrain than wheeled robots

Disadvantages

- Movement speed
- Complexity - actuators and structure are a lot more complicated
- Harder to control - must consider balance and stability
- Less energy efficient due to:
  - Terrain
  - Centre of gravity moves while walking
  - Picking up the legs

Question 3

DOF stands for **d**egrees **o**f **f**reedom, and its defined by the number of joins in each leg. To have a leg that only moves forwards and backwards, it would have two joints: this is because its limited to doing a lift and swing motion. To move backwards it just swings in the other direction than normal. Most robot legs have three joints.

Question 4

Our formula to find how many states there are is $2^k = 2^4 = 16$ states, so the fact that Figure 1 has 16 states reinforces its validity.

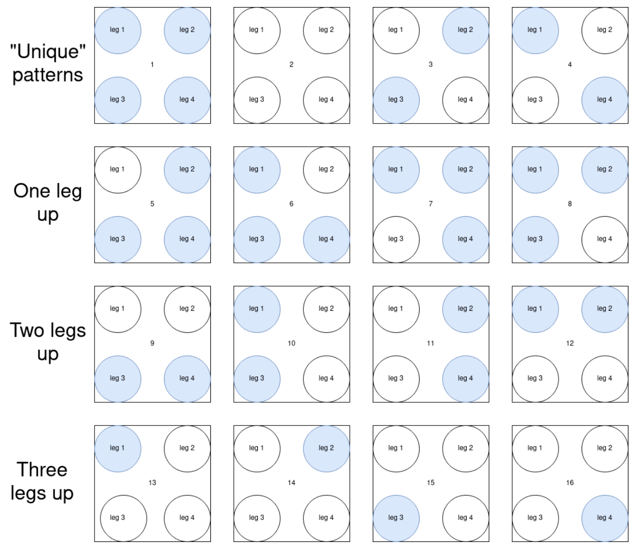

Figure 1: Blue means a leg is down and white is the leg is up

The gaits of a four legged robot is depicted in figure 2.

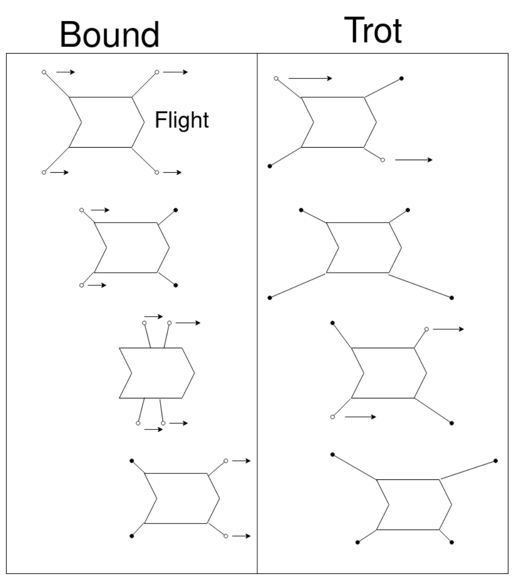

Figure 2: The gaits of a 4 legged robot

Question 5

Note that the ways to represent STRIPS varies from source to source, and the course content talks about the domain model. Considering the variance in the representations of STRIPS, it was decided that doing the way that the course content represents a domain model would be the best way. In the end, these precise details about how the notation is done should not be a major concern as long as it can be understood.

Predicates

- Location(?X) : X is a location
- BoxLocation(?X) : The box is at X
- Bananas(?X): X is bananas
- MonkeyLocation(?X) : Monkey is at X; can be a location or a box (given the monkey is on the box).
- Have(?X) : Monkey has X
- MonkeyOnBox(?X) : Monkey is on the box at location X

States

- <u>Initial State</u> Location(A), Location(B), Location(C), Box(box), Bananas(bananas), BoxLocation(C), At(bananas, B), MonkeyLocation(A)
- <u>Goal state</u> Have(bananas)

World Operators

- MoveToLocation(?X, ?Y)
  - Pre: Location(?X), Location(?Y), MonkeyLocation(?X)
  - Effects+: MonkeyLocation(?Y)
  - Effects-: MonkeyLocation(?X)
- MoveBox(?X, ?Y)
  - Pre: Location(?X), Location(?Y), MonkeyLocation(?X), BoxLocation(?X)
  - Effects+: BoxLocation(?Y), MonkeyLocation(?Y)
  - Effects+: BoxLocation(?X), MonkeyLocation(?X)
- ClimbOnBox(?X)
  - Pre: Location(?X), BoxLocation(?X), MonkeyLocation(?X), MonkeyOnBox(FALSE)
  - Effects+: MonkeyOnBox(?X)
  - Effects-: MonkeyLocation(?X) // this is to avoid situations where the monkey is on the box and tries to move. MonkeyLocation is technically set to null. We also don't need to program when the monkey climbs off the box as that's not necessary for its goal.
- PickUpBanana(?X, ?Y)
  - Pre: Banana(?X), Location(?Y), MonkeyLocation(?Y), MonkeyOnBox(?Y)
  - Effects+: Have(?X)
  - Effects-:

Question 6

There are two perspectives on whether subsumption has good modularity. Firstly, its granted hea subsumption is a form of the reactive paradigm and its main architecture revolves around modules, it can be considered as highly modular. This is especially visible as modules are layered and reflects parts of object oriented programming as they have the ability to override, subsume and contain one another (when they're layered). Only thing that's arguably missing is inheritance, but that can probably be simulated by layering as well.

The counter argument to this is that its very rarely taken advantage of in practice when subsumption is used [Tang, 2017]. Overall, one would say there's more argument that its modularity is high as it's capable of many OOP properties - even if it's rarely used.

The niche targetability is high as we can design the system in a very specific way depending on the situation. For instance, it can compile down onto a programmable-array logic circuit [Tang, 2017].

For portability, due to the fact that the system operates in layers if we tried to adjust an existing robot to a different application chances are that it would take too much effort to do. As in, if there's a fundamental behaviour in layer 0, or even the second last layer that we want to change it could be a lot of effort. On top of this, adapting to picking different actions based on stimulus is challenging For this reason its portability is quite low, which is different from a reactive paradigm. [Murphy, 2000]

Overall its robustness to failure is arguable. Sensors failing is usually detected, and can be countered by having more than than the minimum number of sensors. Furthermore, the lower layers will continue to operate when higher ones are added, however, the problem arises at high layers. These can only inhibit and suppress lower layers as they actively interfere with them, and therefore they can cause undetected errors in the robot. Due to this its robustness is typically evaluated as low because there is no inbuilt method to identify these failures. [Murphy, 2000] [ShippensburgUniversity, 2007]

In summary, it's modularity is good since it has many OOP functions and the niche targetability is high as it can be programmed on very low level systems. It's main weakness out of the four is its ease of portability to different uses, and its robustness to failure is generally considered low.

Question 7

Basically the planning is done on its own, and the sensing and acting are done together. That's to say that information, sensed or cognitive, is put through the plan paradigm and that gives us directives. These directives allow our sense-act section to decide what to do when it senses information. Its 'hybrid-ness' is visible through this, as it introduces planning from other architectures into a reactive architecture's sensing and acting. This assists with the downside of the subsumption model having difficulty adapting different actions to senses that are similar. The specific way P, SA acts is visible in figure 3.

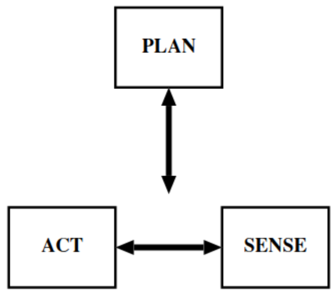

Figure 3: The plan, sense act model for hybrid architecture

The sensing organisation also merges the hierarchical and reactive styles as both planning and acting requires the sensory information. In the model this causes it to be more complex than the hierarchical one, our global world model can have its own sensor and the behaviours can selectively listen to sensors while being able to create virtual sensors. An example is visible in figure 4.

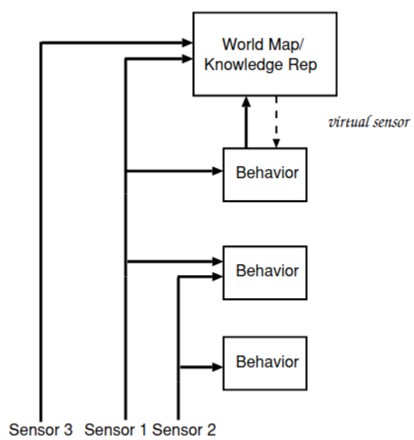

Figure 4: Sensing organisation of hybrid paradigm. Taken from [Murphy, 2000]

Question 8

Shakey's initial model, made in the 1960's was designed using a hierarchical paradigm and the second model slightly changed this [Kuipers et al., 2017]. As such, this question will be split into two sections due to the comparison being different between both of these.

First Model - 1966

The overwhelming issue with the first verson of shakey is the fact that it effectively went blind when it was not in the sensing stage of its plan-sensing-acting paradigm, which would happen to be majority of the time [Murphy, 2000]. This is due to both the hardware on board (SDS 940 computer [Kuipers et al., 2017]) and that the STRIPS paradigm is slow. For reference, a SDS 940 computer has a whopping maximum main of 64 kilowords and is old enough that its processing speed was measured by microseconds to add numbers, not hertz.

To try to equate this to modern hardware, it takes 77 microseconds on the SDS 940 to do a 24 bit floating operation. In other words it has (1/0.000077 = 12987 FLOPS(24 bit)(floating pointer operations per second)) [Systems, 1965]. On the other hand, an AMD Ryzen Threadripper 3990X has a whopping 13,209GFLOPS, or 13209000000000 FLOPS (32bit) [Chiappetta, 2020]. This massive difference will allow a modern Shakey to spend next to no time in the planning stage.

However, in terms of the planning algorithmic speed, the hybrid paradigm has O(n) algorithms [Murphy, 2000] whereas STRIPS planning varies depending on the situation, but will always be PSPACE-complete; that is, it will never exceed $O(n^k)$ in complexity. This is because hybrid models only plan up to the next stage, whereas a hierarchical model plans everything from beginning to end, runs a small section of it, then replans - even if it was correct [Nilsson, 1984].

However, even if we manage to keep up with STRIPS's much higher demand for processing, the integral problem with it still exists: its single threaded. A hybrid can see a situation, plan it, and while its acting if it goes wrong it can replan to react to the situation. On the other hand, STRIPS would finish acting out its plan then realise something went wrong. An example is that say Shakey is running to the bathroom, but a person walks in front of it - a hybrid would instantly be able to react, while Shakey would finish its plan (driving into you for a long time) and then realise something is wrong afterwards. For this reason, the STRIPS model would have to be changed to compete with hybrid robots.

Second Model - 1971

In the second model, Shakey was upgraded to a PDP-10 and due to this processing improvement they changed the model into a layer hierarchical architecture. The distinguishing feature of this is its capability to actually see while acting, but the fact that its a five layered hierarchical system means it came with five times the processing demand [Nilsson, 1984]. In terms of big-O notation, this isn't actually a visible change as the number of layers is just multiplied by some constant ($O(cn^k) \to O(n^k)$. Good thing this is comparing it to modern hardware which is billions of times faster.

Depending on the task, the revised Shakey would be equally as functional as a hybrid robot. That's to say on an especially simple task it would be able to complete it like a hybrid, but if a hybrid model even uses one percent of the processor, the layered hierarchical model likely will not be able to handle that. If we define capability by the bleeding edge functionalities of robots, and the majority of those would use the entire processor if it can, then it is not as capable as a hybrid model.

To sum it up, the first version fails to be as capable as a hybrid due to its vision and time complexity while the revised version would not be as capable because of the time complexity difference between linear and PSPACE-complete's worst case.

References

[Chiappetta, 2020] Chiappetta, M. (2020). Amd threadripper 3990x review: A 64-core multi-threaded beast unleashed. https://web.archive.org/web/20200318200417/https://hothardware.com/reviews/amd-ryzen-threadripper-3990x-cpu-review?page=3.

[Kuipers et al., 2017] Kuipers, B., Feigenbaum, E. A., Hart, P. E., and Nilsson, N. J. (2017). Shakey: From conception to history. https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/General%20Essays/Shakey-aimag-17.pdf.

[Murphy, 2000] Murphy, R. R. (2000). Introduction to ai robotics. https://www.researchgate.net/file.PostFileLoader.html?id=53ad1225d4c118ad118b4705&assetKey=AS%3A272118967668741%401441889625514.

[Nilsson, 1984] Nilsson, N. J. (1984). Shakey the robot. https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/shakey-the-robot.pdf.

[ShippensburgUniversity, 2007] ShippensburgUniversity (2007). Reactive paradigms basics. https://web.cs.ship.edu/~djmoon/roboteam/robotics-notes/murphyc4.pdf.

[Systems, 1965] Systems, S. D. (1965). Sds 940 time-sharing computer system.

[Tang, 2017] Tang, F. (2017). Cs512: Robotics. https://www.cpp.edu/~ftang/courses/CS521/notes/reactive%20architecture.pdf.