
Bachelor Thesis

# Decentralized vs Centralized Controllers For Optimized Mobility In Modular Robots

by

**Baris Dursun**

(2689180)

July 5, 2023

# Decentralized vs Centralized Controllers For Optimized Mobility In Modular Robots

Baris Dursun
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
b.dursun@student.vu.nl

## ABSTRACT

Typically when training models for robot locomotion there is a centralized control system, meaning that there is a central part of the agent that makes the decisions concerning the movement of the agent. But this policy does not translate well to agents of different morphologies, considering the difference in shape, size, and movable segments in individual agents. One policy that does have the capacity to adapt to abrupt changes is a decentralized control system. In this paper, we will explore how a decentralized control system trained in the same way as a centralized one compares in regard to the performance of a gecko-shaped agent. For the decentralized controller, each segment of the agent will have a network that communicates with its 3 closest neighbors and produces an output that translates into the movement of the concerned joint. This model will be optimized using an evolutionary algorithm. The performance of this model will be compared to a centralized control system that uses the same network structure and optimization process. In the end we observe that the decentralized control system produces results on par with its centralized counterpart when it comes to locomotion.

## 1 INTRODUCTION

Robotic systems are becoming more and more relevant as time passes, be it in household usage, industrial means, or scientific exploration, robotic systems have increased and gained popularity in recent years. Currently, most robotic systems use centralized control systems to operate, meaning that there is a control center in one part of the robot, which gathers information from various sensors in the body to decide the next movement of the agent. While this is a good policy for a singular robotic system, it reduces the versatility of the models by a significant margin. Unlike a natural language processing model, which can be pre-trained and then tweaked a bit to adapt to different inputs[2], if the robotic model is trained to control a specific morphology, let's take a robot shaped like a human for example, then it will not be able to adapt to a robot shaped like a spider. The control system will fail to adapt to the new shape, causing the robot to have a declined performance in locomotion. And if robotic technologies keep increasing as such, we will need models that can adapt to different circumstances. A possible solution to this is to create decentralized control systems, where each movable segment of the agent will make its own decisions based on the info it gets from its neighbors. The segments will all have their own network and at each time point, they will pass and receive signals from the other joints and produce an output based on that information. Other papers have already explored the potential of these kinds of models adapting to different kinds of morphologies where it was shown that these types of decentralized systems have the capacity to adapt to different types of morphologies and consistently show above-average results when it comes to locomotion.[1] But would a decentralized control system be able to yield results on par with a centralized control system under similar parameters and optimizations? This paper focuses on comparing a centralized and a decentralized control system trained using the same methods to see if the decentralized system can yield results comparable to the centralized one.

## 2 CONTROL SYSTEMS

### 2.1 Simulation

For this research, The MuJoCo advanced physics simulator is used in combination with Python's Revolve2 library. For each simulation, there is one agent present in the environment. In this research, our agent is a gecko-shaped modular robot with 6 movable joints and 7 segments. The parts that produce the actual movement are the joints. To be able to move, the robot requires a targeted degree of movement for each time step. each joint has the capability of moving 90 degrees but if the joints are given the possibility to move in their full range of motion each frame, after their optimization, they will move to their full extent in one time-step, making their velocity too much and breaking the rules of the physics engine. Here the solution proposed to fix this issue was to limit their range of motion for each frame. So they still have the capability to move 90 degrees but they can just move 1 degree per frame. This will be discussed again in the optimization and the future work segments.

**Figure 1: Morphology of the agent used in this experiment**

### 2.2 Network

Linear neural networks are commonly used in robot locomotion tasks, but they are generally used in combination with reinforcement learning methods[1][3][4]. However feed-forward linear neural networks are usually used for pattern recognition tasks, meaning that they have no information about what happened in the previous time-steps. Because of the setup of our simulation, as it was discussed before, we can not give large degrees of movement to

each joint, so we can not rely on pattern recognition for locomotion. For this reason, a recurrent neural network is a much more suitable model for our purposes, since it uses recurrent computation to capture the sequential dependencies in the data. In this paper, we aim to use a recurrent neural network trained by evolutionary algorithms to obtain optimal results. The network used for this task is a recurrent neural network with one recurrent layer and one linear layer for mapping the hidden state to the output dimension. The network uses a hyperbolic tangent function(tanh) as its activation function. As input, we use data gathered from the joints. At each time-point, we look at each joint and gather the joint position x,y,z; the joint orientation as x,y,z,w; the body's position that the joint controls as x,y,z; the body's orientation as x,y,z,w and the body's center of mass as x,y,z. So in total, we have 17 data points per joint and 6 joints per agent. That gives us 102 usable data points per step of the simulation. In the centralized model, we give all this information as the input so the input size is 102 and the output size is 6. One targeted degree of movement for each joint. In the decentralized model, we take the information of the joint itself and three of its closest neighbors. So we take 68 input values and produce 1 output value, the degree of movement for the joint itself. There is also a slight randomness implemented in the input where a value between -0.1 and 0.1 is added to the values in the input tensor. This is there to solve a freezing problem that arises in the simulation caused by the model giving the same outputs as the inputs. In certain cases, the output of the simulation will lead to the same targeted degree of movement as the agent's previous position, making the agent freeze in place permanently. For example, if the output of a certain joint is 0, then the joint will not move, causing the input data to be the same and in consequence, the output of the next time-step to also be zero, effectively soft-locking the agent in place. The implementation of slight randomness in the input values fixes that problem, as well as adds some noise to the gathered data.

| Networks | Inputs | Outputs |
|---|---|---|
| Decentralized controller network | 102 | 6 |
| Centralized controller network | 68 | 1 |

**Table 1: Input and output numbers of the networks**

## 2.3 Centralized Control System

The centralized control system takes in all 102 data points, transforms them into a tensor, and presents this tensor as a single input sequence. This is fed into the network with the initial hidden state of size 200. The initial hidden state is a tensor of zeros. After the initialization of the network, the recurrent layer processes the input, updates the hidden sequence, and produces the output. The output is then passed through the linear layer to map the hidden step representation to the desired output dimension. The values obtained at the end of this process are the targeted degrees of movement for the next time step. For this model, there is no communication between neighbors, since the model already knows the positional information for each joint. This process is repeated for each time step to obtain the desired degrees of movement.

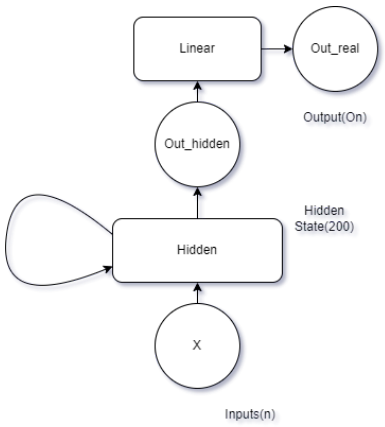


**Figure 2: The structural representation of the network used for the control systems**

## 2.4 Decentralized Control System

The agent used in this paper has 6 joints. In the system, it is represented as a tree structure with each joint presented as a node. Please see Figure 3 for a graphical representation. Each joint of the agent, so each node of the tree communicates with 3 of its closest neighbors. So let us take joint number 1. It will first look at its own positional and orientational data, and then the data of its 3 closest neighbors, joints 2, 3, and 4. With each joint having 17 data points as information, joint 1 will take 68 inputs. After this, it will pass its inputs to the decentralized recurrent neural network. The process after this is quite similar to the centralized recurrent neural network. Again a hidden state of zeros of size 200 is created and both the hidden state and the input tensor are passed to the recurrent layer. The hidden state is updated and an output is produced which is mapped to the desired output dimension through a linear layer. In the end, a single value is obtained as the output. This value is the degree of movement joint 1 should perform in the next time step. This process is repeated for each joint until all the joints have a targeted degree of movement. Then this movement is applied to the agent and the process begins again with the new input that was obtained after the movement. If the joint has more than 3 neighbors in equal distances as the closest neighbors, it will pick 3 of them and start the process by communicating with them. In the end, this leads to every joint only being able to control its own movement while seeing the positional information of its neighbors.

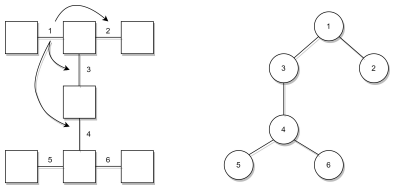


**Figure 3: A representation of the communication between the joints of the agent.**

## 2.5 Optimization

An evolutionary algorithm is used to optimize these models. First, a model list of the desired size is created. This list contains n randomly created models, n being the population size. This will be our initial population. After this, each model is put in a simulation of 100 seconds. The distance traveled by each agent is recorded to be used as the fitness values for each of the models. After all the simulations are done, we look at the fitness values and choose the n/2 best models. We call these models the best model list. We add these models to our next-generation list. After this, we perform mutation on these models and add the result to the next generation as well. For the mutation operation we take the best-performing models and we add random values between -0.5 and 0.5 to each weight and bias in the network. We create one mutated model for each member in the best models list, so the mutate function gives us population size / 2 as well. We add these models to the next generation too. When we put all these models together in the next generation list, we obtain a list equal in size to the population list we used in the first batch of simulations. Then we change the population list with the next generation and rerun the simulation for how many generations we want to simulate. Throughout this process, we store the fitness data of the best model in each generation to analyze the improvement in the fitness of the models over time. In the earlier iterations of this algorithm, there was also a crossover function and a random model generator. For the crossover operation, we first picked Parent 1 and Parent 2. Then we swapped the weights and the biases of the two models so that we would have two new models with the weight of Parent 1 and the bias of Parent 2, and vice versa. But the crossed-over models did not produce meaningful results and they just increased the run-time of the algorithm. Similarly, there was a random model creator for each generation that created a certain number of new models to add some randomness into our batches, but this too did not produce meaningful results either, and instead of increasing the fitness, both the cross-over and the random models were actively hindering the progress of the optimization. The best results were obtained using only the existing fittest individuals and the mutation function so the final experiments were all done using only those two.

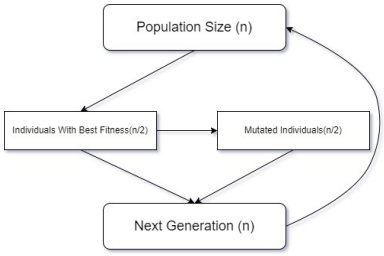


**Figure 4: The evolution process for the optimization.**

# 3 EXPERIMENT SETUP

To optimize the agents, batches of 40 agents, so both the population size and the offspring size of 40, were run for 100 generations. The agents were tested for 100 seconds each on flat surfaces with no obstacles. After 3 generations It was observed that the learning curve was almost flat at the 40 to 50 generation mark, and considering the enormous run time of the algorithm, the generation count was dialed back to 70 generations per training. To see the merged graph of 3 training sequences of 100 generations, please observe Figure 5.

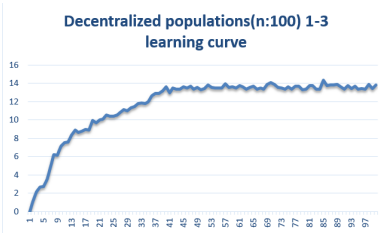


**Figure 5: This plot represents the merged data of 3 training sequences. Each point in the plot is the best fitness functions of each generation for the decentralized optimizer.**

# 4 RESULTS

## 4.1 Evaluation

During the optimization process, the fitness score for the best agent is recorded for each generation. So at the end of each generation, we have a database containing 70 fitness scores, one for each generation. From this database, we can observe the increases in the fitness scores, meaning the learning rate of the agents. We will run the evolutionary algorithm 5 times for both centralized and decentralized models, combine all the fitness scores obtained during the simulation and plot the results to observe the average learning rate per model. To combine the results we put the fitness data of each optimization process side by side and take the average of each row. A visual representation of this process can be found below in Table 2.

| Generation | Fitness 1 | Fitness 2 | Fitness 3 | Combined Fitness |
|---|---|---|---|---|
| 1 | 0.06 | 0.02 | 0.1 | 0.06 |
| 2 | 0.2 | 0.7 | 0.3 | 0.4 |
| 3 | 1.4 | 1.6 | 1.5 | 1.5 |

**Table 2: The method of obtaining the fitness data used in the merged graphs representing the learning rate of the controllers**

To analyze the overall performance of the systems, both the centralized and the decentralized models will be run 10 times in 100-second simulations and their fitness scores will be recorded. Again the average of the fitness scores will be taken to conclude which agent has the highest overall performance. This will give us an overall idea about the performances of the models, but to get significant results, a two-sample independent $t$-test of sample size 100 will be conducted to compare the means of the distance traveled by both models.

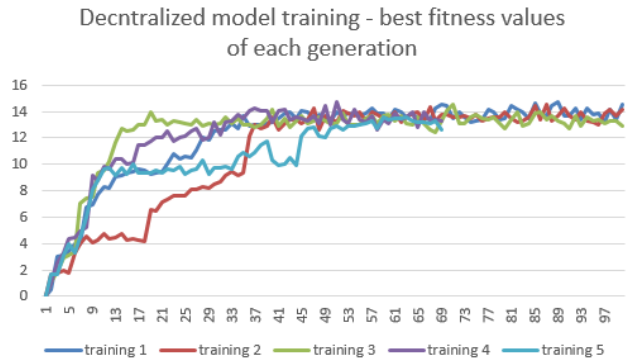


**Figure 6: The best fitness values of each generation for 5 training sequences of the decentralized controller.**

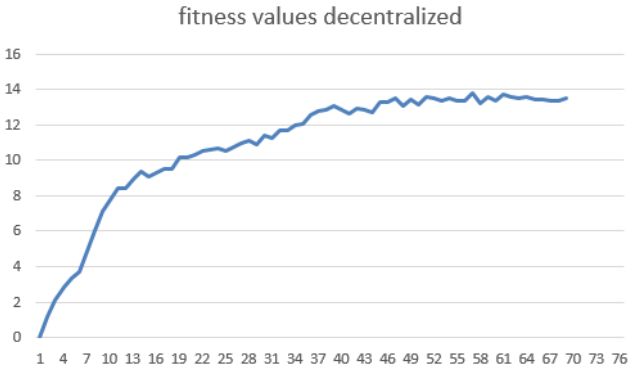


**Figure 7: Averaged out learning rate for the decentralized controller**

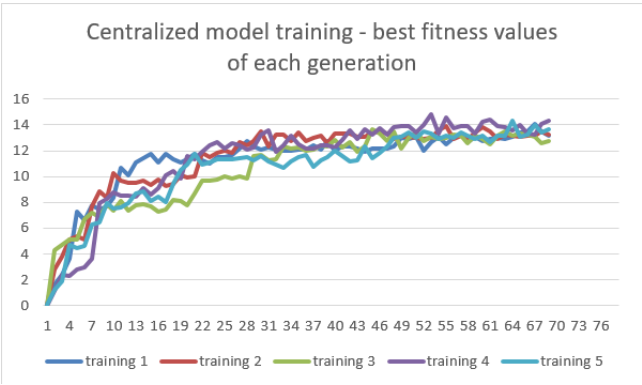


**Figure 8: The best fitness values of each generation for 5 training sequences of the centralized controller.**

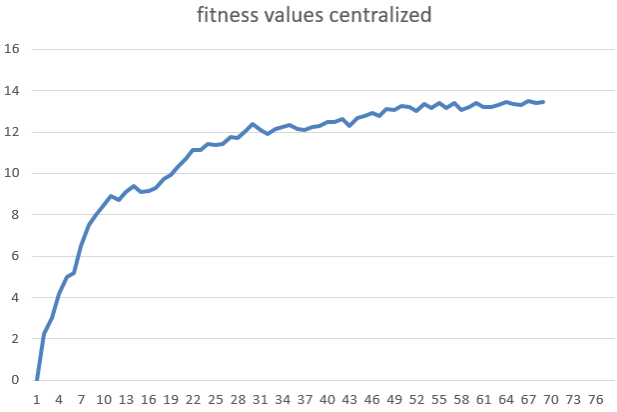


**Figure 9: Averaged out learning rate for the centralized controller**

## 4.2 Learning Rate for the Decentralized Model

After 5 training sequences of the decentralized controller, it is observed that the agents rapidly improve in their locomotion capabilities in the first 10 generations, going from 0 to 8 in their fitness score. After the 10Th generation, the agents keep improving, albeit at a slower pace, until somewhere between the 40 to 50 generation mark. At this point, the curve of the learning rate flattens and the agents stop improving with each passing generation. The learning curves of each 5 training sequences can be seen in Figure 6, and the averaged-out learning rate of the decentralized controller can be seen in Figure 7. After the training, the agents controlled by the decentralized controller are able to traverse a distance of 13 to 14 meters in each simulation of length 100.

## 4.3 Learning Rate for the Centralized Model

Similarly to the decentralized controller, after 5 training sequences of the decentralized controller, it is observed that the agents have a steep improvement in their learning rate in the first 10 generations, going from 0 to 9 in their fitness score. After the 10Th generation, the agents keep improving at a slower pace, although faster than their decentralized counterparts until the 25 to 30 generation mark, where the improvement in their fitness scores slows down and flattens around the 50Th generation mark and the agents stop improving with each passing generation. The learning curves of each 5 training sequences can be seen in Figure 8, and the averaged-out learning rate of the centralized controller can be seen in Figure 9. After the training, the agents controlled by the decentralized controller are able to traverse a distance of 13 to 14 meters in each simulation of length 100 seconds, making their performance in locomotion almost identical to the decentralized controller concerning the distance traveled.

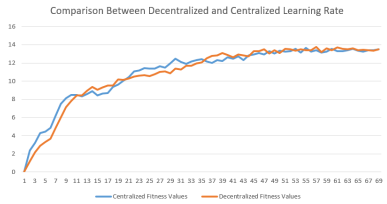


**Figure 10: Comparison between the centralized and the decentralized systems learning rates**

## 4.4 Average Performances of Both Models

To compare the locomotion efficiency of both models, we run both of them for 10 times in 100-second simulations and store the distance traveled. Then we average out the traveled distance to figure out the average distance that the model is able to travel in 100 seconds. The average distance traveled by the centralized controller was 14.2 meters and the average distance traveled by the decentralized controller was 14.6 meters. These findings can be found in Table 3.

| | Centralized controller | Decentralized controller |
|---|---|---|
| Distance Traveled | 14.203576241833492 | 14.584721805737365 |

**Table 3: Average distance traveled at the end of the simulations**

## 4.5 Hypothesis Testing

Comparing the agent's performance by just running them 10 times and averaging the results gives us a vague idea of the locomotion capabilities of the models but it does not statistically prove that the decentralized model have the capability to perform as well as the centralized model. For this a two-sample independent $t$-test was conducted. To gather the samples, we run both of the models for 100 simulations of length 100 and stored the distance traveled at the end of each simulation. A $t$-test was used to compare the means of two samples to see if there is a significant difference.

*Null hypothesis(h0)*: There is no significant difference between the decentralized and the centralized controller when it comes to locomotion efficiency.
*Alternative hypothesis(h1)*: There is a significant difference when it comes to locomotion performance between the two controllers.

After gathering 100 samples for each controller, the mean, median and the standard deviation of the data was computed.

| Data Analysis | Centralized controller | Decentralized controller |
|---|---|---|
| Mean | 13.27 | 11.19 |
| Median | 13.23 | 10.82 |
| SD | 7.99 | 6.82 |

**Table 4: Data analysis of the samples gathered at the end of 100 simulations. The SD stands for standard deviation.**

| | |
|---|---|
| $t$-statistics | 1.22 |
| $p$-value | 0.22 |
| significance level | 0.05 |

For the $t$-test, a significance level of 0,05 was used.

The p-value(0.22) is greater than the significance level of 0.05, so we fail to reject the null hypothesis(h0). It can be concluded that there is not enough evidence to conclude a statistically significant difference between the two groups (decentralized and centralized models) in terms of the distance traveled, demonstrating the fact that the decentralized controller can produce results on the same level as the centralized controller when trained using the same parameters and conditions.

# 5 CONCLUSION

In this work, we built two control systems for a modular agent. The purpose of both systems was to provide optimal locomotion to its host agent. One system was a conventional centralized controller where one control center gathered data from all the limbs and produced movement for all the joints of the agent for the next time-step. The second controller was a decentralized one where each joint of the agent communicated their positional information with their neighbors. Based on this information, each joint produced its own movement for the next time-step. Our question was, can this decentralized controller produce results on par with its centralized counterpart? After looking at the fitness scores of the training sequences for each of the models, we can say that the learning rate and the best distance achieved by both of the systems were quite similar, flattening around 40 to 50 generation mark and producing fitness scores between 13 and 14 meters, with some outliers in-between. After comparing the performances of the trained agents and hypothesis testing for our claims, we can conclude that the decentralized system can consistently produce results on par with the centralized system.

# 6 DISCUSSION

At the start of this research, the main hypothesis was that the decentralized controller would not be able to yield results on the same level as the centralized controller. The reason for this being that the main strength of decentralized controllers are their adaptability to different circumstances, be it different segment sizes of the agents, a different environment than the original training environment, or a different morphological structure than the agent used for training. This makes the decentralized controller a great template for being able to pre-train models intended to be used in robot locomotion. But in this research the adaptability aspect of the decentralized controller was never used. Instead the controller was trained and tested on the same agent, on the same environment, using the same parameters. All the points where conventional centralized controllers excel at. Although it is still worth noting that in the samples used for hypothesis testing, the decentralized system did perform marginally worse than its centralized counterpart. This difference was not enough for us to reject the null hypothesis, but if the sample sizes were bigger, the difference between the performances may increase as well. The borderline is that even though the decentralized controller used in this research did produce results on par with the

centralized controller, the best way of furthering the research on robot locomotion would be to use the decentralized controllers for pre-training large models and using centralized models to fine-tune the pre-trained systems to achieve optimal locomotion in shorter time frames, combining the best aspects of both simulations.

### 6.1 A Small Remark About the Learning Rate of the Agents

In the fitness graphs for the training sequences such as Figure 6 and Figure 8, some dips in the fitness scores were observed, especially after the 40+ generation mark. This is also the time frame where the improvement rate of the models with each generation flattens out. The decrease in performance is due to the randomness factor in the input. The robots do not move exactly the same with each iteration of the simulation even if they have the exact same starting position because of the slight randomness factor in the input data. After the 40Th generation mark, the models stop improving and the same best models start being generated over and over again, so the randomness factor in their movement may cause them to produce results less optimal compared to their previous run. But these dips in performance are usually negligible and change small amounts concerning the performance of the agent. An agent that is able to traverse 14 meters may just traverse 13 meters in another simulation, but for the duration of all the training sequences, the deviation in performance of the same models has never fallen more than 2 meters.

## 7 FUTURE WORK

For the future development of these models, removing the randomness factor from the inputs would allow the agents to produce more consistent results that would be easier to analyze. The small random additions to the inputs are there to fix the freezing problem of the agents that arise when the network produces an output that will make the subsequent joint stay at the same position that it was in the previous time step. A function that prevents these kinds of outputs may lead to better results concerning the performance of the agents than introducing slight randomness to the data.

Apart from this, the random neighbor selection can be excluded from the decentralized controller. Instead of selecting random 3 neighbors when the join has more than 3 neighbors, each joint can select all the possible closest neighbors, so all the neighbors closer than 2 steps in the tree representation and if the joint does not have enough neighbors, the empty slots in the input can be filled with 0's. So let's say every joint has 4 neighbors apart from one that has 3. Instead of choosing 3 random neighbors for each joint, we can choose 4 neighbors for each joint and fill the missing information in the input of the 3 neighbored joints with 0's.

In this research, only one type of agent was used, to effectively compare the performances of the centralized and the decentralized systems. So the decentralized system that was made for this research does not have the capability to adapt to a different agent if the agent has a different number of joints than 6. Implementing permutation invariance into the model so the network can perform with different numbers of inputs, meaning different numbers of joints can make the controller much more adaptable, especially towards agents with different morphologies.

The cross-over function used in this research caused diminished returns when it came to the learning rate of the agents, and subsequently excluded from the optimization process. But a better cross-over function, one that maybe instead of swapping the weights and biases between the parents, merges the weights and the biases, may lead to an improved learning rate, hence, and improved performance.

To keep the agents from moving too fast and breaking the velocity barrier of the simulation, we clamp the output of the network so that the concerned joint can only move one degree per time point. However instead of clamping the output, getting the output as is and realizing the movement in parts in each time frame might produce better results. So instead of giving the joints at most one degree of movement per step, we can get the actual movement and then divide the movement into smaller movements to feed into the agent in the following time steps. This may produce more accurate movement while traversing the simulation.

Lastly doing the hypothesis testing with bigger sample sizes may lead to more accurate results. The sample sizes used in this research were kept to 100 samples each due to time constraints. But in hindsight this is a low number for hypothesis testing, and repeating the tests with a larger sample size may lead to more accurate results.

## REFERENCES


[1] Wenlong Huang, Igor Mordatch, and Deepak Pathak. 2020. One Policy to Control Them All: Shared Modular Policies for Agent-Agnostic Control. In *ICML*.

[2] Kenton Lee Jacob Devlin, Ming-Wei Chang and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. (2019).

[3] Srinivasan Sriram Jay Lemmon Josh Merel Greg Wayne Yuval Tassa Tom Erez Ziyu Wang S. M. Ali Eslami Martin Riedmiller David Silver Nicolas Heess, Dhruva TB. 2017. Emergence of Locomotion Behaviours in Rich Environments. (2017).

[4] Trevor Darrell Pieter Abbeel Sergey Levine, Chelsea Finn. 2016. End-to-End Training of Deep Visuomotor Policies. (2016).