# Precise Positioning of a Drone using Spoken Language Commands

**Florian Langer**

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
*Master of Philosophy*
in Machine Learning and Machine Intelligence

Sidney Sussex College 20th August 2020

## Declaration

I, Florian Langer of Sidney Sussex College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.
Standard Python libraries were used for implementing the algorithms discussed in this report, including PyTorch as the deep learning framework of choice. This report contains a total of 11, 437 words.

[illegible] 20/08/20

Florian Langer Date

## Abstract

This project aims at learning spatial representations of spoken position descriptions for indoor robotics. Our work is a crucial component of fully autonomous indoor assistants and household robots that interact with humans by listening to their instructions.

Despite its huge importance for many robotics applications, to-date to the best of our knowledge there exists no work which explicitly learns the mapping from spoken instructions to 3D-positions. The work most related to ours is in the field of scene rendering from text description. Here some work has started to explore learning relative positions of various objects with respect to each other given a natural language description. While this resembles our approach we focus on learning precise positions in 3D space rather than relative positions between objects. Other approaches try to extract spatial relation from natural language by mapping text into formal symbolic language where individual symbols have predefined relative positions associated with them. However, by constraining themselves to a set of predefined relations the accuracy and the richness of positions that can be described are very limited.

In order to predict a target position we train a neural network to predict Gaussian mixture model parameters of a 3-dimensional probability distribution over the position. We train this model on a dataset containing target positions around a reference cube and the corresponding natural language description. By training our model object agnostic we are able to learn meaningful distributions using just 200 training examples. We evaluate our model in the simulation of a real room. Here we demonstrate that by combining the probability distributions obtained from several descriptions of the same position we can increase the accuracy in our distribution. When providing up to 5 descriptions we show that we can pinpoint positions in a room to an accuracy within 50 cm. Having evaluated the system in isolation we directly use the position predictions to navigate a simulated drone to the predicted target positions. Finally we have tested our approach on a small commercial drone and show that our system can be successfully transferred to the real world.

# Contents

## List of Figures

## List of Tables

# 1 Introduction

## 1.1 Objective

In this thesis we address the problem of positioning a drone using a spoken language instruction. We use a drone for this project as it is highly mobile and easy to control. However, all insights we make can be readily transferred to other robots. The main challenge in maneuvering a drone by telling it where to fly lies in estimating a position in space from a natural language description. In our case an example of a typical position description is something akin to "Over the table, slightly to the left and further back". This description references an object and specifies the desired position with respect to it. By training a model on similar descriptions we learn a mapping between spoken descriptions and 3-dimensional probability distributions over the target position. These probability distributions can then be used by robotic systems to perform a range of different tasks. In directly applying the predictions from our model to navigate a real drone we bridge the gap between a theoretical procedure to a real world implementation.

## 1.2 Motivation

Many realistic scenarios involving indoor assistants and household or factory robots involve a human instructing a robot. For a human the easiest form of communicating its intentions is by simply telling the robot in natural language what to do, for example “Put the milk into the fridge next to the juice bottle” or “Clear the floor underneath the table”. While simple for humans these are very challenging tasks for a robot to perform as they require a range of highly involved skills, ranging from speech recognition, natural language processing and reasoning to accurate localisation and object detection. Crucially the described system does not just understand what to do but also where to do it. For this it needs to understand the spatial information that is contained within the prepositions in combination with reference objects (“underneath the table”,“next to the juice”). When trying to hard code these relations one quickly finds that any rule based system will be highly in accurate and quickly overwhelmed by a large number of “edge cases”. Ultimately, as so many intricacies of this world, relative positions have to be learned, rather than predefined.

## 1.3 Challenges

The key challenges for understanding semantic descriptions of positions in space are the following:

**Intrinsic Imprecision:** Inferring a position in 3D-space from a natural language description is difficult due to the intrinsic imprecision in the description. Taking for example a common preposition such as ”to the right” to refer to a position with respect to some other object, we realise that this description usually describes a wide range of positions. While humans are fairly good at narrowing down the intended position from the context of the situation and a knowledge of the objects present, even they struggle. This can be seen in the fact that often when humans describe to each other how to position themselves, they make their best attempt and then ask a question such as “Like this?” or “Here?” to double check if this position was indeed intended.

**Ambiguity:** Apart from all position descriptions being intrinsically imprecise many descriptions are also ambiguous. This means that the description does not just describe a range of positions in close vicinity of each other, but positions that are at completely different places in space. An example of such is “Stand next to the corner of the table.” which as most tables have four corners could imply any four different positions.

**Viewpoint Dependency:** Another obvious challenge when learning spatial relations is the dependency on the viewpoint. This could be the viewpoint of the speaker, the viewpoint of the robot or any other viewpoint described by the speaker. However, sometimes this task is made more challenging when the speaker changes its viewpoint without explicitly saying so. Take for example a human facing a car from the side. The description “a cow is in front of the car” could now mean the cow is between the human and the car (using the humans viewpoint) or the cow is next to the car [1] (when viewed from the humans viewpoint but in front of the car when viewed from the cars viewpoint)[1].

**Arbitrariness:** Furthermore even for a perfectly symmetric, isolated box in an empty room humans struggle to categorise different spatial prepositions into different regions in 3D-space [2]. This is because borders between positions such as ”behind the box” and ”left of the box” are always fluid and somewhat arbitrary.

## 1.4 Approach

In order to allow a drone to successfully fly to a described position it needs to infer the precise coordinates of the target position, know its current location and given the two know

[1]The latter in this case is actually the more probable intended meaning as a car has a clear front.

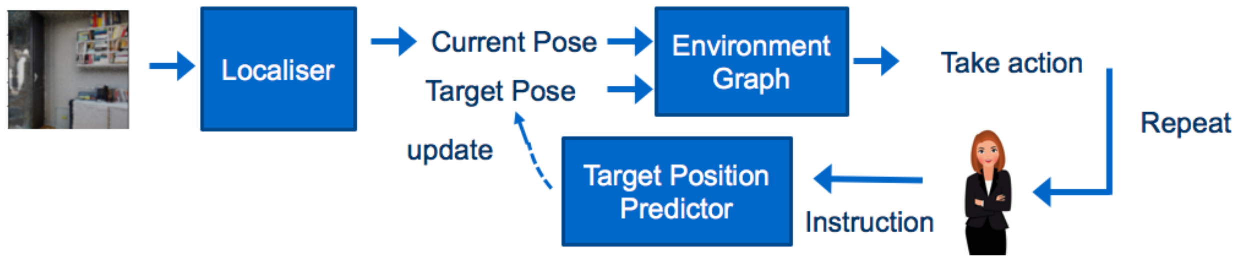


Figure 1: Schematic of the different modules interacting in our system. We use a CNN to estimate the current pose. Simultaneously we predict a distribution over the target position from a natural language description. We take the maximum of this distribution[2] and use it in combination with the current estimated pose to query a graph of the environment for the next action that should be taken. We then take this action and repeat the process until we have arrived at the desired target position. In order to refine the target position a human can add further target descriptions at any time.

what action it should take next. Figure 1 depicts how the three components *Target Position Predictor*, *Localiser* and *Environment Graph* which we use for each of these tasks interact to achieve the overall goal.

**Target Position Predictor:** The challenges presented above have enforced us in our conviction that a true solution to relative positioning has to be learned rather than predefined. We therefore follow an end-to-end deep learning approach which goes straight from a position description to a 3-dimensional probability distribution over positions. To do so we have collected our own dataset of target positions in the vicinity of a unit cube and corresponding text description. The cube serves as a placeholder object whose dimensions at test time are replaced by those of the referenced object approximated as a cuboid. During training our network learns to predict probability distributions around the object. At test time the predicted distributions are used to calculated the probabilities on a global grid. The node with the highest probability is selected as the estimated target. If multiple descriptions are specified we sequentially update the global grid by simply adding the probability contributions from the individual descriptions[3].

**Localiser:** In order to localise in space we train a ResNet based localiser on images rendered inside a 3D model of our testing room in Blender[3]. Here we directly regress the 4-DoF camera pose (containing the 3 dimensional position as well as the orientation with respect to the z-axis). Usually a pose is described by two additional rotational parameters

[3]This means that the "probability" for a given node in the global grid can in fact be greater than 1 it is therefore better to think of them as a general score.

that encode the rotation around the x and the y-axis. However, for our purposes we limit ourselves to the orientation around the z-axis as for a slowly moving drone in stable flight the other two are fixed.

**Environment Graph:** For navigation we use a graph of our environment. This graph consists of nodes corresponding to the positions on a regular 3-dimensional with 20 cm intervals. For each position we create 12 nodes corresponding to the different possible orientations in 30° steps. Nodes are connected to its nearest neighbours in this 4-dimensional grid by directed edges labeled by the action required to move from the starting node to the final node. Nodes that fall within objects are disconnected from its neighbours. In this way all valid paths in the graph avoid the objects present in the scene. In order to use this graph to determine what first action should be taken to move from the current pose to the target pose we perform the following steps: First, we find the nodes closest to the starting pose and the target pose. We then find the shortest path between the two nodes. Finally we retrieve the action that corresponds to the first edge in our shortest path and return this action as the final recommendation.

## 1.5 Contribution

The three main contributions of this thesis are the following:

- We investigate a novel approach for inferring positions from their natural language description by directly learning probability distributions from a collection of labelled data. To the best of our knowledge there is no existing work following a comparable approach.
- We have created a new dataset containing an image of a scene containing a reference object and a small cube visualising the desired target pose in combination with the precise scene coordinates as well as the corresponding text description. This dataset will be made available to the scientific community for future research.
- In demonstrating the feasibility of our approach we open a new avenue of future research which can build on our work by using a larger number of training examples and more complex models.

## 1.6 Outline

This thesis is divided into 6 sections, the first of which is this introduction. Section 2 gives an overview of work that is related to this project. Section 3 explains in depth the approach we follow to tackle the problem of positioning a drone using spoken instructions. It explains the different models we are using and how they interact to achieve the overall goal. This is followed by details of the training procedures we have used for training those models in Section 4. Section 5 demonstrates the range of experiments that were conducted before arriving at the final system architecture. Finally we give concluding remarks in Section 6 and point out the potential for future work.

# 2 Related Work

This section briefly describes existing work which is related to ours. Of great importance for testing our system in simulations and the real world is a functioning localisation procedure. The existing work which we make use of is described in 2.1. In 2.2 and 2.3 we briefly touch upon previous work on encoding spatial natural language and using it for scene rendering. Finally in 2.4 we discuss the general field of language-based navigation.

## 2.1 Localisation

There exist three different approaches for tackling the image localisation problem: image retrieval and key point matching, direct camera pose regression using CNNs and scene coordinate regression.

### 2.1.1 Image Retrieval and Key Point Matching

The first class of localisation methods compares key points in a query image to the key points of all images in a database to find a set of closely matching candidate images. For these candidate images the scene coordinates for the key points matching with the query image are retrieved from sparse 3D models of the local scenes. These scene coordinates of the matched pixels are then used to estimate the 6-DoF camera pose [4]. While this method is accurate for images in textured environments, it can fail for less textured ones as an insufficient number of key points may be found for reliable matching.

### 2.1.2 Direct Camera Pose Regression

Another class of localisation methods ([5],[6]) uses CNNs to directly regress the 6-DoF camera pose from singe RGB images. The outputs of the network consist of the absolute camera position $\mathbf{x}$ and the orientation parameterized as a quarternion $\mathbf{q}$. The network is trained by minimizing the Euclidean loss

$$\text{loss}(I) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2 + \beta \left\| \hat{\mathbf{q}} - \frac{\mathbf{q}}{\|\mathbf{q}\|} \right\|_2 . \tag{1}$$

Methods based on direct pose regression are very fast at test time as they only require a single forward pass through the network. While they work better for untextured environments than approaches relying on key point matching, they are often very sensitive to environmental changes as it is difficult to simulate them during training. Despite this drawback for the simplicity of the training procedure we will use a CNN for directly regressing the camera pose in this project.

### 2.1.3 Scene Coordinate Regression

A final class of localisation methods does not use a CNN to directly regress a camera pose, but rather to first predict per pixel scene coordinates. In a second step a pool of pose hypotheses are calculated from small subsets of predicted scene coordinates. For each of the pose hypotheses a score function evaluates the consensus with all other predicted scene coordinates. The pose prediction with the maximum score is then iteratively refined and finally used as the estimated camera pose[7]. As pose hypothesis are sampled from small subsets of scene coordinates they can successfully predict camera poses despite considerable environmental changes. However, the biggest drawback of approaches using scene coordinate regression are long training times. They also have not been shown to scale yet to large indoor environments.

## 2.2 Encoding Spatial Natural Language

To this point most efforts in this field have focused on extracting spatial relations by mapping text into formal symbolic languages [8]. While numerous annotation schemes and methods for doing so have been proposed ([9],[10]), none of these offer the advantages of learned representations. Similarly, work on visualising spatial descriptions has relied almost exclusively on heavily hand engineered representations [11],[12]. In order to make use of the advantages of learning representations in this project we devise an end-to-end deep learning approach in which we go straight from a semantic position description to a 3-dimensional probability distribution over a target position.

## 2.3 Scene Rendering from Language Descriptions.

Even though the problem of rendering 3D scenes from language description is different to the problem of predicting a distribution over a target position from a language description, both of them share the need for encoding spatial relations from natural language. It is therefore worth having a look at existing work in this field. So far there have been quite a few approaches to convert language into a 3D scene. However, just as the approaches for encoding spatial language directly most of these are heavily rule based [14],[15] making them unpractical for interpreting actual speech. Recent work by *Ramalho et al.*[13] learns to render simple scenes containing primitive objects from text descriptions alone. They generate both synthetic language descriptions consisting of pair wise descriptions between objects as well

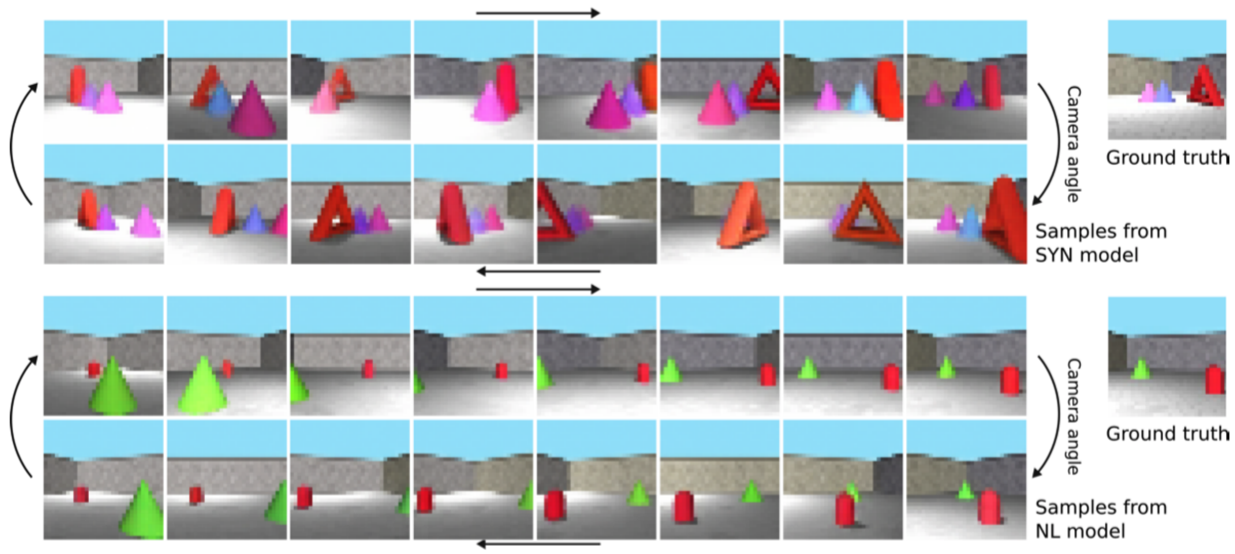


Figure 2: Taken from [13] : Sample outputs obtained for synthetic (top) and natural (bottom) language scene descriptions. The respective descriptions are: "There is a pink cone to the left of a red torus. There is a pink cone close to a purple cone. The cone is to the left of the cone. There is a red torus to the right of a purple cone."; "There are two objects in the image. In the back left corner is a light green cone, about half the height of the wall. On the right side of the image is a bright red capsule. It is about the same height as the cone, but it is more forward in the plane of the image."

as natural language descriptions. While results for synthetic instructions are a lot better, they do show successes for natural instructions too (Figure 2).

## 2.4 Language Based Navigation

The ability to command robots in complex environments using natural language has been a research goal for many decades [16]. While earlier work often avoided using vision cues by relying on explicit labels for navigation goals or certain object, in recent years more and research has focused on explicitly combining these. This seems crucial as for true mastering of language based navigation, language and vision have to be intricately linked. Not only is this task interesting from a research perspective as it promises insights into general questions on multimodal representations but it also paves the way for real world applications, such as personal assistants and in-house robots [17]. Classical tasks here are room-to-room navigation based on a semantic description of the route by a human. While this is a challenging task, the focus here lies on following the described path rather than reaching a very accurate final position. If the target position is reached within a 2 m radius the trial is considered a success. However, of course this is still a very inaccurate position and shows the need for our work. To a certain extent our work can be seen as a more fine-grained version of language based navigation. One difference of our work though is that rather than describing a precise trajectory we focus on describing just the final target position.

# 3 Method

In order to position a drone with voice commands by describing a target position three main steps are necessary. As a first step we have to predict the implied position based on the verbal instruction we receive. As a second step we have to localise in space as only if we know our current position can we predict what actions we should take as a third step. We outline how we achieve each of these task in the following.

## 3.1 Speech to Position Conversion

When trying to translate speech into a position in space it quickly becomes apparent that a scalable and robust solution has to be learned rather than can be predefined. The reason for this is that natural language, and especially spoken language, is simply too complex and variable to process with a rule-based system. Additionally the intrinsic ambiguity in descriptions of positions which also depend on the viewpoint of the speaker and many unspoken assumptions add another layer of complexity to the problem. The key contribution of this project lies in a learned procedure for predicting a 3-dimensional probability over space for a target position given a semantic description of the position with respect to a reference object.

### 3.1.1 Data Acquisition

In order to collect data for training a target predictor we create a cube of unit scale in Blender. We then sample a position on a $9 \times 9 \times 9$ grid around the cube where the distance between neighboring nodes is half the side length of the cube. For three selected training examples the position of the target is visualized as a small green cube in Figure 3. We use further visualisations of the scene to fully understand the position and then describe it with respect to the cube from a fixed view point. An example of such a description is “Behind the cube, but on the right side and higher up”. While in general position descriptions with spoken language always have a certain uncertainty associated with them, they can also imply inherently different positions, e.g. the “Next to the corner of the cube” may imply any one of 8 different positions. While our model is well capable with dealing with such ambiguous descriptions our training data only consists of position descriptions that imply a single position. We transcribe the speech input to text using the Microsoft speech API [18]. The creation of one training pair including time for thinking of an appropriate description and speaking takes about 30s. In this manner we collect 250 pairs of positions with respect to the cube and corresponding target descriptions. After the data collection we correct for errors in the transcription by listening to the recordings of sentences which were transcribed falsely.

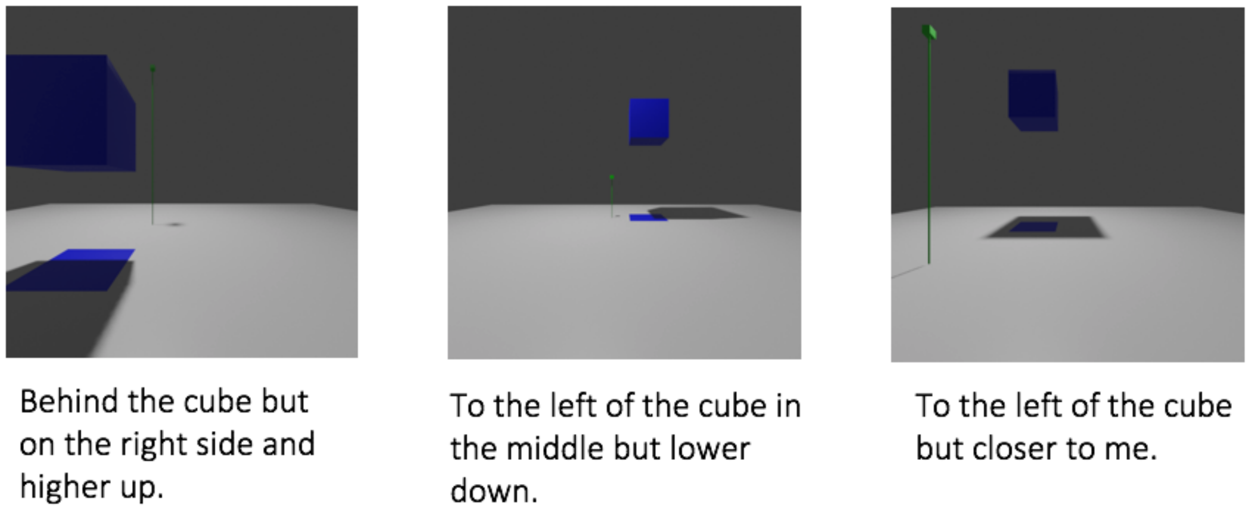

Behind the cube but on the right side and higher up.
To the left of the cube in the middle but lower down.
To the left of the cube but closer to me.

Figure 3: Examples of position visualisations (green cube) and their semantic description with respect to a reference cube (blue).

### 3.1.2 Model

Our prediction model comprises of three components: a fastText word vector model [19], a Long Short-Term Memory cell [20] and a Mixture Density Network [21].

1. **fastText embedding**
   The fastText word vector model embeds individual words into a 300-dimensional feature space. It was trained as a Skipgram-model on a huge text corpus during which it has learned to embed similar words as similar feature representations. Its network weights are kept constant in our training procedure.

2. **LSTM Cell**
   We sequentially feed the word embeddings we receive from the fastText word vector model through an LSTM. In this way we maintain information about the order of the words in the sentence description while at the same time allowing the network to learn which words in the description are most relevant. We use the hidden representation we receive after feeding all $N$ word embeddings through the LSTM as an encoding of our description which is passed on to the Mixture Density Network.

3. **Mixture Density Network**
   The Mixture Density Network uses the encoded description to predict a multi-modal Gaussian probability distribution over positions in space. It consists of a Categorical

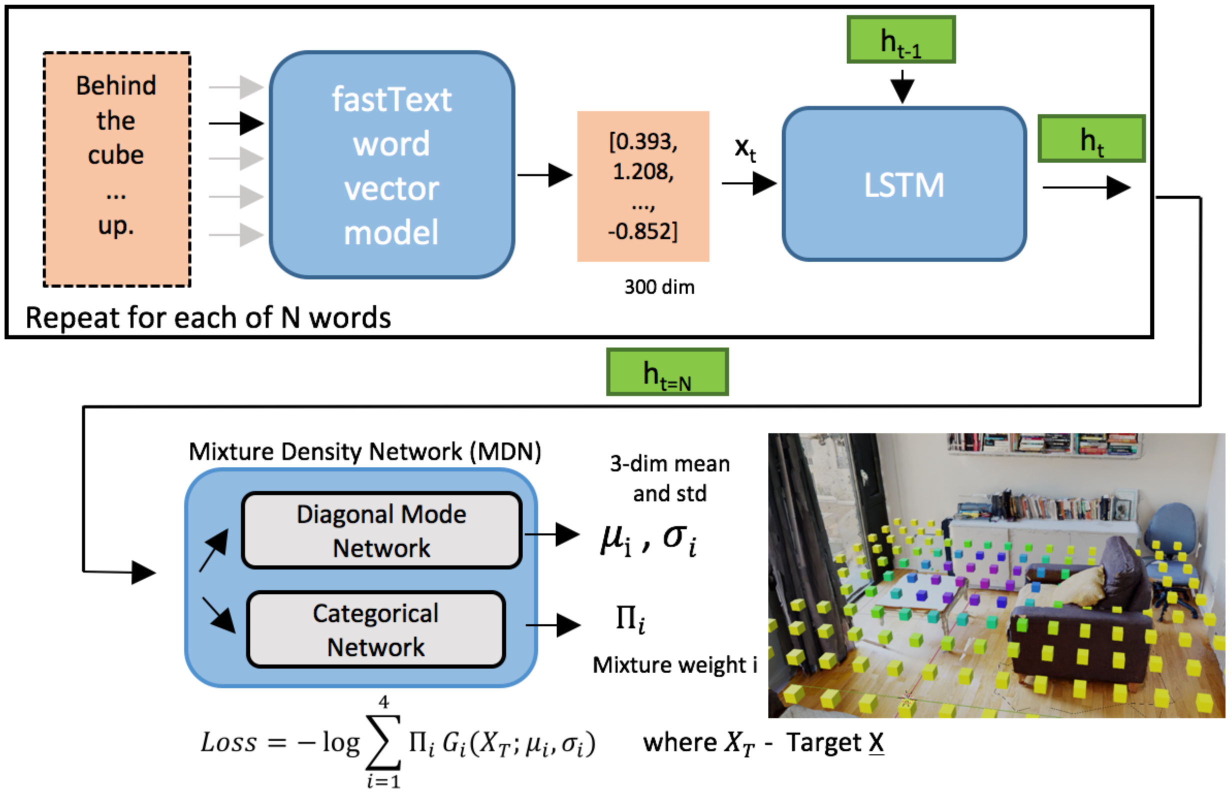


Figure 4: Target Predictor Model: After receiving a target position description we embed its words individually into a 300 dimensional feature space using a fastText word vector model. These embeddings are then sequentially used as an input to an LSTM cell. The final hidden representation containing information about all $N$ words in the description is passed as an input to a Mixture Density Network[21]. Here one component predicts mean $\mu_i$ and the diagonal standard deviation $\sigma_i$ for each mode in the target distribution while a second component predicts mixture weights $\Pi_i$. The network is trained end-to-end by minimising the negative log likelihood of the target positions. Note that the weights of the fastText word vector model are kept fixed during the training procedure.

Network for predicting mixture components $\Pi_i$ and a Diagonal Mode network for predicting 3-dimensional mean and standard deviation $\mu_i$ and $\sigma_i$ for each mode $i$. For our experiments we choose to use 4 modes. This would allows us to predict truly multi-modal distributions corresponding to ambiguous descriptions such as "next to". Furthermore even for learning single mode distributions it may be helpful to allow for multiple modes as this may avoid predicting extreme standard deviations.
The Categorical Network consists of a single 300-dimensional hidden layer with ELU activation function given by

$$ELU(x) = max(0, x) + min(0, (exp(x) - 1)) \tag{2}$$

and outputs the log probabilities for the individual modes which are then exponentiated and normalised to 1.
Similarly the Mode Network has a single hidden layer with 300-nodes and uses an ELU activation function. It outputs the 3-dimensional mean and the logarithm of the standard deviation for each of the mode components. The loss used to minimise during training is given by the negative log likelihood,

$$Loss = -\log P = -\sum_{i=1}^{4} \Pi_i \mathcal{G}(\mathbf{x}_{\text{Target}}; \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i) = -\sum_{i=1}^{4} \Pi_i (2\pi)^{-\frac{3}{2}} \det(\boldsymbol{\Sigma}_i)^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)} \tag{3}$$

where $\boldsymbol{\Sigma}_i = \text{diagonal}(\sigma_{i1}, \sigma_{i2}, \sigma_{i3})$.

### 3.1.3 Test Time

The reason why all our training data is provided just for an abstract cube rather than physical objects is that by describing positions object agnostic we are quickly able to collect sufficient amount of training data to apply our system in real world scenarios with many different objects. We choose a cube rather than some other primitive object like a sphere or a cylinder as many objects especially in indoor environments have rectangular shapes.
At test time we use a map of our environment which contains the important objects parameterised as cuboids. In order to make a prediction we search the text input for a known object in the room and replace it by the word "cube" as our network was trained on position descriptions using the word "cube". The mean of the predictions the networks make for each mode are then shifted to the center of the anchor object $\mathbf{x}^{\text{object}}$ and the standard deviations are scaled by the object dimensions $\mathbf{s}^{\text{object}}$. Therefore the final probability distribution over

the target position $\mathbf{x}_{\text{Target}}$ reads as

$$P(\mathbf{x}_{\text{Target}}) = \sum_{i=1}^{4} \Pi_i \mathcal{G}(\mathbf{x}_{\text{Target}}; \boldsymbol{\mu}_i^{\text{pred}} + \mathbf{x}^{\text{object}}, \boldsymbol{\sigma}_i^{\text{pred}} \odot \mathbf{s}^{\text{object}}) \,, \tag{4}$$

where $\odot$ implies element-wise multiplication.
In order to ensure that target positions can be predicted precisely we allow a user to describe a single position with multiple descriptions. For each description we predict the mixture density parameters and combine them with the parameters of the referenced anchor object to calculate the probabilities for each node on a tight 3D world grid. We repeat this for each description and then simply add the probabilities on the world grid. This makes our procedure robust to individual miss-predictions and allows for greater precision in the final position.

## 3.2 Localisation

A crucial component in our system is a reliable procedure for localisation. Continuous localisation becomes necessary as the model for the dynamics of the drone is complex. Air currents and imprecision when executing commands cause the drone to deviate from the intended path. Small imprecision in orientation and position quickly accumulate to significant errors and have to be mitigated for by continuous localisation. This stands in contrast to most ground robots which by tracking the revolutions of their wheels can calculate their current position precisely, and makes our problem more challenging.
In order to develop a cheap and scalable solution we rely solely on single camera images. While using more advanced sensing, such as depth sensors, simplifies the problem of localisation (and particularly object avoidance), the drones that are equipped with such are more expensive. We attack the problem of localisation by training a deep neural network with a Resnet18 backbone to directly regress the 4-DoF pose $(x, y, z, \theta)$ from a single RGB-image.

### 3.2.1 Data Acquisition

In order to collect data for training our model we follow two different approaches:

- **Matterport 3D Scanner**
  For the first approach we use a dense 3D model with textures of our target rooms. This model was created using the Matterport Pro 2 Camera. The Matterport Pro 2 creates panoramic images by rotating around a fixed tripod. Images taken from different position, in combination with their depth information captured by infrared

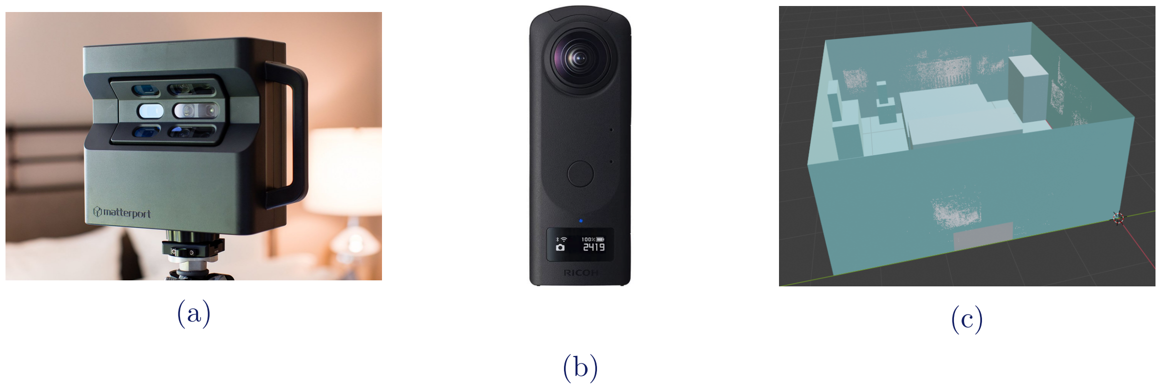


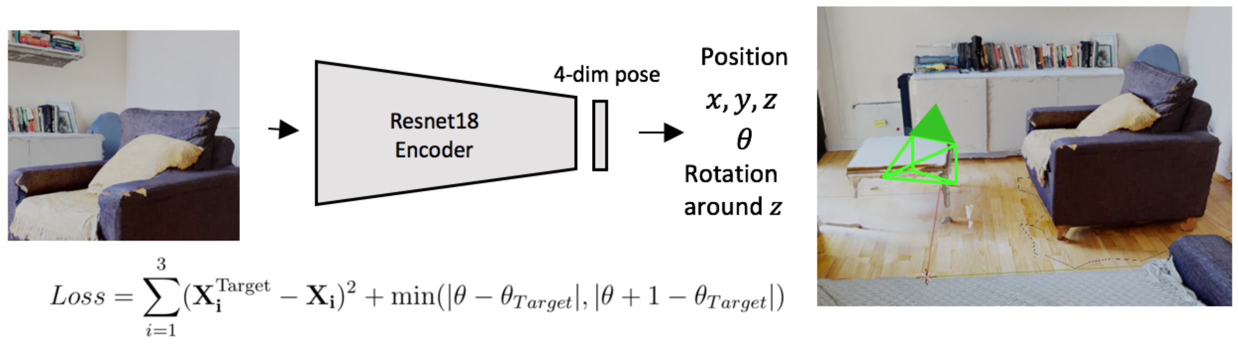
Figure 5: a) Matterport Pro 2 and b) Theta 360. c) We estimate the global transformation for converting points in the sfm coordinate system obtained from images of the Theta 360 to our room coordinates by aligning reconstructed points with primitive objects in our room.

sensors, are automatically combined to textured 3D meshes of the environment. We load these meshes into Blender and render 60.000 images corresponding to random camera positions and random orientations around the vertical. These images are used for training the localisation network we use for testing our system in the simulated test room.

- **Ricoh Theta 360 camera**
  For the second approach we use a Ricoh Theta 360 camera and create a video of traversing a grid in our test rooms. While we can combine the images from the video into a point-cloud using structure-from-motion, the point-cloud we get is not dense enough to render directly from it (as we are able to do with the model obtained with the Matterport Pro 2). We therefore simply use structure-from-motion to determine the world coordinates and orientation of the camera for each frame in the video. By aligning the predicted point cloud from the structure of motion algorithm with primitive models of the objects in our room (see Figure 5 c) we find the global transformation between the two coordinate systems. We then crop smaller images with a horizontal field of view of 55 degree (corresponding to that of the drone we are using for final evaluation) from the 360 images and use these as our training data for our pose network.

### 3.2.2 Model Architecture

For localisation we train a convolutional neural network which uses a ResNet18 based encoder to encode our images into a 512-dimensional feature representation. We pass these features through 2 fully connected layers with 128 and 4 nodes respectively. A ReLU activation



$$Loss = \sum_{i=1}^{3}(\mathbf{X}_\mathbf{i}^{\text{Target}} - \mathbf{X}_\mathbf{i})^2 + \min(|\theta - \theta_{Target}|, |\theta + 1 - \theta_{Target}|)$$

Figure 6: Localisation Network: We use a deep neural network with a ResNet18 backbone to directly regress the 4-DoF poses from single RGB-images. The 3-D scene coordinates of the position of the camera are used directly whereas the orientation around the $z$-axis is normalised to the range from 0 to 1. The loss is given by the Euclidean loss for the position plus a linear loss contribution from the orientation which takes the cyclic property of angles into account.

function is applied to all hidden layers. The output of the last layer is made up of the room coordinates of the pose $\mathbf{X} = (x, y, z)$ as well as the orientation around the vertical $(\theta)$ normalised to the interval between 0 and 1. We only need a single number to describe the orientation of our drone as we only take pictures when hovering. This means that roll and pitch of the drone are fixed and only yaw, which is the rotation around the z-axis changes. We use just this single angle rather than all 3 Euler angles (or any other parameterisation of the orientation, e.g. quarternions) as this simplifies the learning procedure for the network as distinct features will always be detected at similar orientations.

When training the network we apply a Euclidean loss to the position and a linear loss to the normalised orientation angle (taking into account that angles are cyclic) such that our loss is given by

$$\begin{aligned} Loss &= \alpha\, L_{pos} + (1-\alpha)\, L_{ang} \\ &= \alpha \sum_{i=1}^{3}(\mathbf{X}_\mathbf{i}^{\text{Target}} - \mathbf{X}_\mathbf{i})^2 + (1-\alpha)\min(|\theta - \theta_{Target}|, |\theta + 1 - \theta_{Target}|) \end{aligned}$$

where we choose $\alpha = \frac{1}{2}$ empirically.

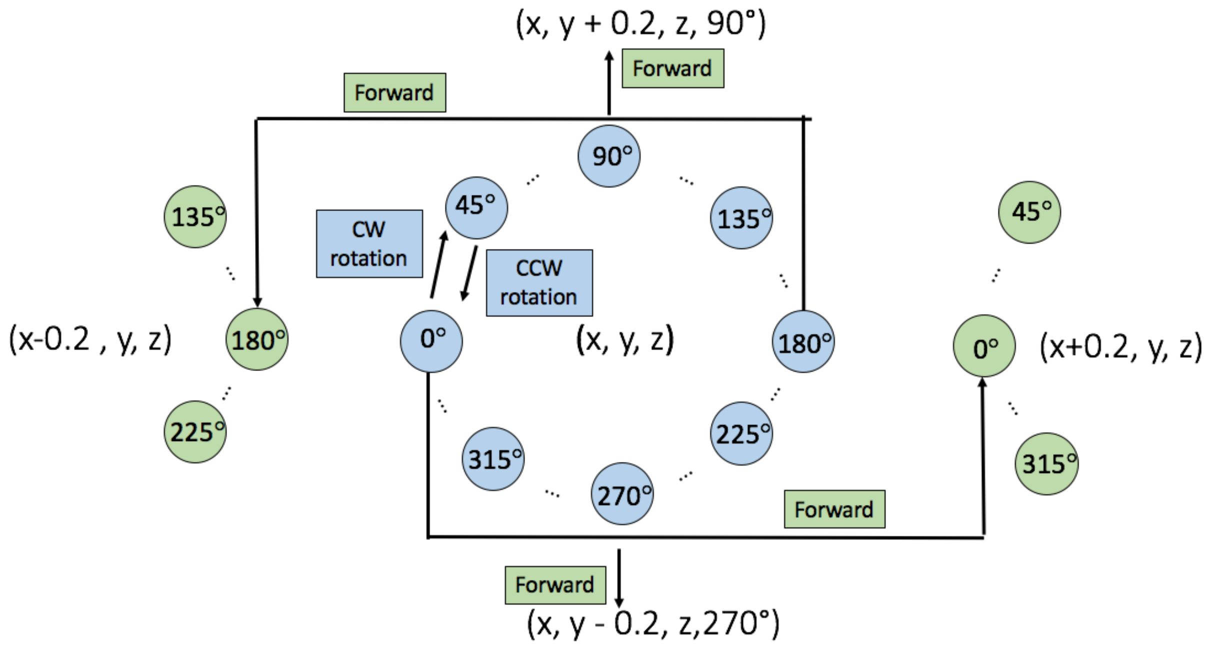


Figure 7: Top-down view of a small extract of the Environment Graph used for navigation: Nodes in blue all correspond to the same position in space with different orientations. Rotations (blue) connect these nodes to their nearest neighbours. A "Forward" action (green) connects neighbouring positions at the same height with each other for those nodes whose orientation is aligned with the direction of the translation in space.

## 3.3 Navigation

For navigation purposes we use a bi-directional graph of our environment (see Figure 7). This graph is constructed as follows: We discretise the parameterised room into a set of evenly spaced nodes separated by 20 cm in the $x$, $y$ and $z$ direction from neighbouring nodes. For each position in space we create 12 nodes corresponding to 12 different orientations around the vertical axis in 30° steps. Nodes at the same position are connected to their neighbours with one unit of rotation more or less by edges. Neighbouring positions are only connected for those nodes that have the correct orientation which corresponds to the translation between the two positions. This ensures that the drone will always face the way it is flying which is desirable for most real-world applications as it unsafe to fly in a direction from which we do not receive a live image from. In the horizontal we also only allow for movements along the 4 principal direction (i.e. positive and negative x-direction and positive and negative y-direction) as this prevents the graph from getting too big leading to overly long search times. In the vertical nodes are connected to their equivalent one layer higher and one layer lower. Each edge is labelled by the action that is required to fly from its starting node to

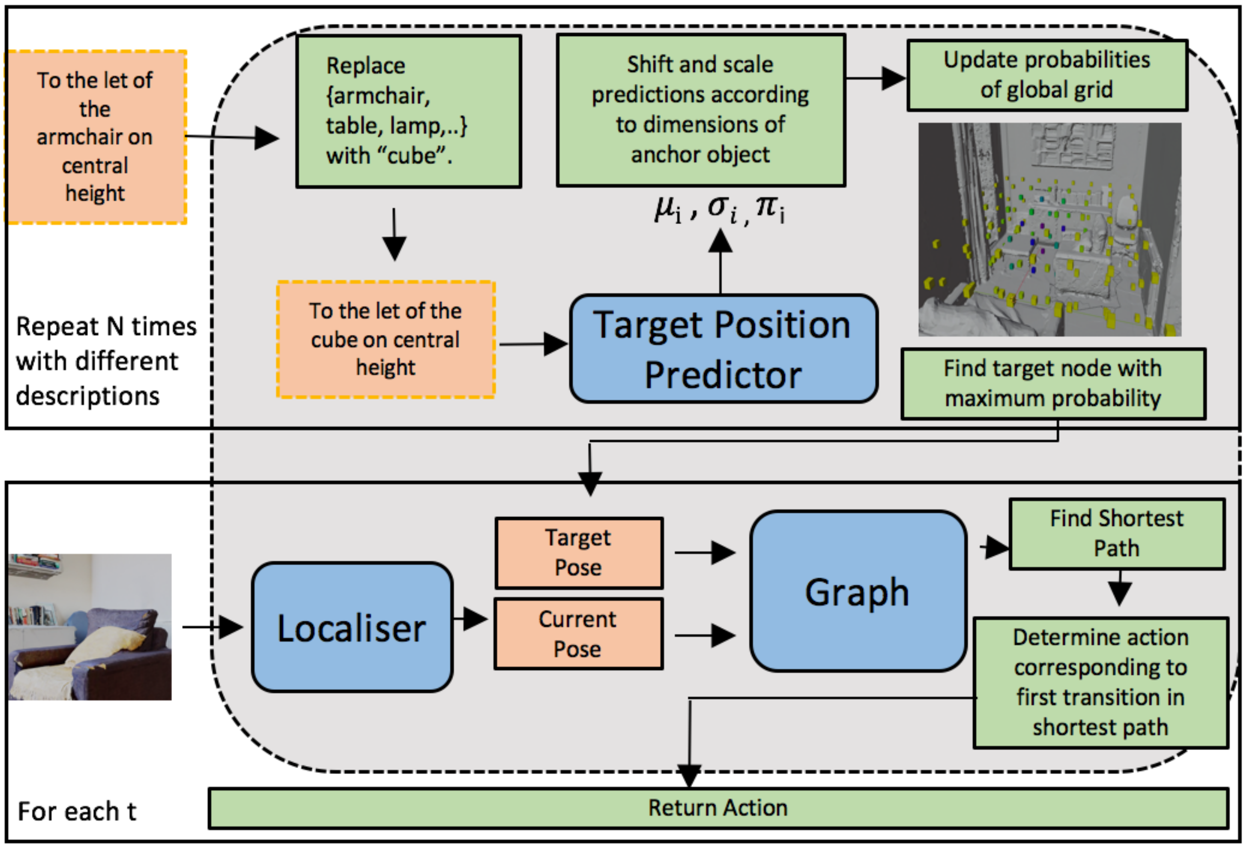

To the let of the armchair on central height
Replace {armchair, table, lamp,..} with "cube".
Shift and scale predictions according to dimensions of anchor object
Update probabilities of global grid
$\mu_i, \sigma_i, \pi_i$
To the let of the cube on central height
Target Position Predictor
Find target node with maximum probability
Repeat N times with different descriptions
Target Pose
Current Pose
Localiser
Graph
Find Shortest Path
Determine action corresponding to first transition in shortest path
For each t
Return Action

Figure 8: Visualisation of the interplay of the different components in our system to determine what action the drone should take in order to reach a described target position.

the final node. When using this graph for navigation between a starting and a target pose we first find the nearest nodes in the graph. We then use Dijkstra's algorithm [22] to find a shortest path between these two nodes. This shortest path contains a sequence of nodes along it which are connected by edges labelled with the actions required to follow it. We follow those actions until we localise in space again and repeat the process.

## 3.4 Full System

Figure 8 shows how the different components described above come together to predict what actions a drone should take to navigate to a described target position. Starting from the top let corner a position description is recorded and transcribed into text. It is searched for an anchor object to which it refers which is stored and whose word is replaced by the word "cube" in the description. The description is then inputted into the Target Position Predictor which returns Gaussian mixture model parameters for the probability distribution over the target position. The predicted means are shifted to the center of the reference object and the standard deviations are scaled by the dimensions of the cuboid approximating the reference object. Using these transformed parameters the probability contribution to the global grid are calculated. The position of the node in the updated global grid with the highest probability is returned. This process is repeated $N$ times with different position descriptions which in turn update the global probabilities and lead to changing target positions[4]. Simultaneously at each time step we localise by inputting the real or simulated drone view into our localisation network which returns the predicted pose. Together with the predicted target pose we query the graph of the environment for the shortest path between the nodes closest to the two poses. We find the action corresponding to the first edge in the shortest path and execute it after which the process is repeated.

[4]Target positions are simply converted to target poses by appending a fixed orientation

# 4 Experimental Setup

The solution we propose for a drone to understand the descriptions of positions in a room and to successfully navigate to them requires training several neural networks on different datasets. This section gives a concise overview over these datasets and details on the training procedures of the networks. It is concluded by introducing key metrics which we use to assess the performance of the networks in Section 5.

## 4.1 Datasets

We use two kinds of datasets for training our networks: image-pose pairs for training our localisation networks and position-description pairs for training the Target Position Predictor.

### 4.1.1 Localisation

For localisation we have used two datasets. One of them is from a living room for which we had a dense textured mesh which was obtained with the Matterport Pro 2. The section of the living room that is used for experiments is 5.0 m long, 3.2 m wide and 2.4 m high. It contains 10 labelled objects for which we approximated bounding cuboids in Blender. We use the textured mesh of this room to render images corresponding to $60,000$ randomly sampled poses. We use $50,000$ of these for training and reserve $10,000$ for validation. The images are 100 pixel in width and height and were rendered with a camera of $50°$ FoV in the horizontal and vertical direction.

The second dataset we use for localisation was obtained from $360°$ videos that were shot using the Ricoh Theta 360. We took 4 videos corresponding to the 4 main orientations[5] of a bedroom which we traversed in a grid at 3 different heights. We use structure-from-motion to compute the camera position and orientation for every 50th frame in the video. This leaves us with 600 images from all 4 main orientations from which we crop $128 \times 128$ images in $3°$ intervals between $-45°$ and $+45°$ around the center. This ensures that the person holding the camera is never in the crops. In this manner we generate $18,000$ images and their corresponding poses of which $16,000$ are used for training and $2,000$ for validation. During training we make small random adjustments to brightness, contrast and saturation to avoid overfitting. Similarly we perform slight random zooming of our images in order to simulate small imprecision in our procedure of determining the FoV of the drone.

[5]Even though we use a $360°$ camera the orientation still matters as parts of the images will be occluded by the person taking the video

### 4.1.2 Position Descriptions

We use Blender to randomly sample 250 positions around a unit cube. These positions are then described by a user with respect to the cube. We transcribe the audio files using the Microsoft Speech Recognition API. Errors in the transcriptions are manually corrected for. 200 position-description pairs are used for training, while 50 are reserved for validation.

## 4.2 Network Training Details

This subsection provides the details of the training procedures for the localisation network and the target predictor network.

### 4.2.1 Localisation Network

We train the localisation network in the simulated room by randomly batching the $50,000$ images into mini-batches of size 25. We use an Adam [23] optimiser with learning rate 0.0001. The network is trained for 950 epochs at which point we observe an increase in the validation loss and stop the training. We use the same batch-size, optimiser and learning rate for the localisation network in the real room. As before we employ early stopping and stop the training procedure after 820 epochs.

### 4.2.2 Target Position Predictor

When training the Target Position Predictor we only have 200 position-description pairs and therefore reduce the batch size to 5. Just as for the localisation networks we use the Adam optimiser, here with a learning rate of 0.005. We train the network for 340 epochs until we see an increase in the validation loss and note a deterioration of the predictions for the validation examples.

## 4.3 Evaluation Protocol

In order to evaluate the localisation network at training and test time we compute the Euclidean distance between the position parameters of the predicted pose and the true pose. We also compute the difference in orientation around the vertical axis, taking into account that angles are cyclic. Quantifying the performance of the Target Position Predictor is more challenging and requires the introduction of selected key metrics which will be used throughout Section 5. They are the following:

- **Euclidean-distance from the most likely prediction to the target**
  For this metric we simply find the node on the $9 \times 9 \times 9$ grid around the cube with

the highest probability under the predicted distribution and compute the Euclidean distance to the target,

$$D_{ML}(P_{pred}, \mathbf{X}_T) = \sqrt{(\mathbf{X}_T - \mathbf{X}_{n_{max}})^2} \tag{5}$$

where

$$n_{max} = \underset{n \in \mathcal{N}}{\operatorname{argmax}}\, P(n)\,. \tag{6}$$

Here $\mathcal{N}$ is the set of all nodes in the global grid and $P_{pred}$ is the predicted probability distribution by the Target Position Predictor.

- **Relative probability at the target**
  We define this quantity to be the value of the predicted probability density function at the target normalised by the maximum value of the probability density function,

  $$p_{relative}(P, \mathbf{X}_T) = \frac{P(\mathbf{X}_T)}{P(\mathbf{X}_{n_{max}})}\,. \tag{7}$$

  Therefore rather than evaluating how good the maximum prediction itself is we evaluate how likely the target is under the predicted distribution.

- **Minimum Euclidean distance to the target from the $n$ maximum predictions.**
  Here we consider the $k$ nodes with the highest probability and find the minimum distance of any of these nodes to the target,

  $$D_k(P_{pred}, \mathbf{X}_T) = \min_{n \in \mathcal{N}_{max\,k}} \sqrt{(\mathbf{X}_T - \mathbf{X}_n)^2} \tag{8}$$

  where $\mathcal{N}_{max\,k}$ is the set of $k$ most likely nodes under $P$.

While the first two metrics defined above are important to consider they reduce the predicted probability distributions to single point predictions. This is problematic as we are interested in assessing the probability distribution as a whole. A common way to assess a predicted distribution is to compute the *Kullbach-Leibler* divergence to a target distribution. However, in this case we only have single draws from our target distribution which makes a direct comparison impossible. We therefore introduce the third metric which aims to preserve information about the shape of the distribution.

# 5 Experiments

This section gives an overview over the most important experiments that were conducted for this report. The first set of experiments focuses on the model for converting speech to a position. The results for these will be analysed in detail as they constitute the key novel contribution. The second set of experiments details how the different subsystems outlined above perform together for full navigation to a verbally described position in a simulated room. Finally we show our results when testing our system in the real world.

## 5.1 Speech to Position Conversion

We have investigated various potential loss functions as well as network architectures before reaching our final model. The results for these will be explained in the following.

### 5.1.1 Loss Ablation Study

For training our network we investigated 3 different loss functions:

- **Cross-Entropy**
  When considering the speech-to-position problem on a fixed grid, one can treat it as a N-class classification problem where every position description has to be classified as belonging to 1 of the 729 nodes in the 3-dimensional grid. Looking at the problem in this way using categorical cross-entropy appears as a natural loss function,

$$\text{Loss} = -\sum_{n=1}^{729} t_n \log(p_n) \tag{9}$$

  where $t_n$ is a one-hot encoding of the target node and $p_n$ is the output for node $n$.

- **Binary Cross-Entropy**
  One of the dangers of following the approach above is that the prediction obtained for the classification may be overconfident and focused only on a single node which is too narrow for our purposes. We therefore investigate using binary cross-entropy where for each target location we have two nodes in the final layer of our networks which are normalised by a *softmax* function such that they correspond to the probabilities of the location being the target or not,

$$\text{Loss} = -\left(\sum_{n=1}^{729} c_0\, t_n \log(p_{n,0}) + (1-t_n)\log(p_{n,1})\right) . \tag{10}$$

In Equation 10 $c_0$ is a constant which was empirically set to 10 to speed up the training procedure[6]. By training our network on this loss, at test time we allow it to predict multiple target locations with high probability which is crucial for imprecise descriptions of positions.

- **Mixture Density Network**
  While both losses above allow us to train our network, they lack an encoding of spatial proximity i.e. the notion that if two nodes are close to each other their probabilities of being the target are likely similar. However, this property is very crucial for successfully generalising to unseen position descriptions and for producing smooth and meaningful predictions which reflect the uncertainty in many descriptions. We encode this property by assuming that basically all position descriptions can be well approximated by a mixture of Gaussians. Therefore rather than predicting node probabilities directly we predict Gaussian mixture parameters. We train our model on the loss given by Equation 3 which ensures that the probability of obtaining the observed targets is maximised.

Before comparing the networks trained on the different losses quantitatively it is insightful to compare individual predictions for an unseen position description (Figure 9). The main point to note is that while both the model that was trained on the cross-entropy loss (left) as well as the model trained on the binary cross-entropy loss (middle) do have their high probability predictions on the left side of the cube and behind it, they are very scattered and not smooth, which emphasises the need for hard coding the notion of spatial proximity into the model. The mixture density model on the other hand produces more accurate predictions with a reasonable variance. All the information contained in the description ("to the left","further back","on the same height as the back") is reflected in the prediction.

Comparing the 3 different networks quantitatively in terms of the distance of the maximum prediction from the target (see Table 1) we notice that also just considering its maximum prediction mixture density networks outperform networks with individual node predictions trained on cross entropy and binary cross-entropy losses on unseen data. On seen training data the latter two perform significantly better than the mixture density network. However, this simply demonstrates that they over-fit faster on the training data.
Comparing the results for the relative probability at the target one can see that the model trained on binary cross-entropy has higher relative probability at the target than the mixture density network. While this may seem surprising at first, we note that we perform a slightly unfair comparison as for the model that was trained on binary cross-entropy predictions

[6] Otherwise the contribution of the huge number of positions which are not the target dominate initial phases of training and the network simply learns to predict no target at all.

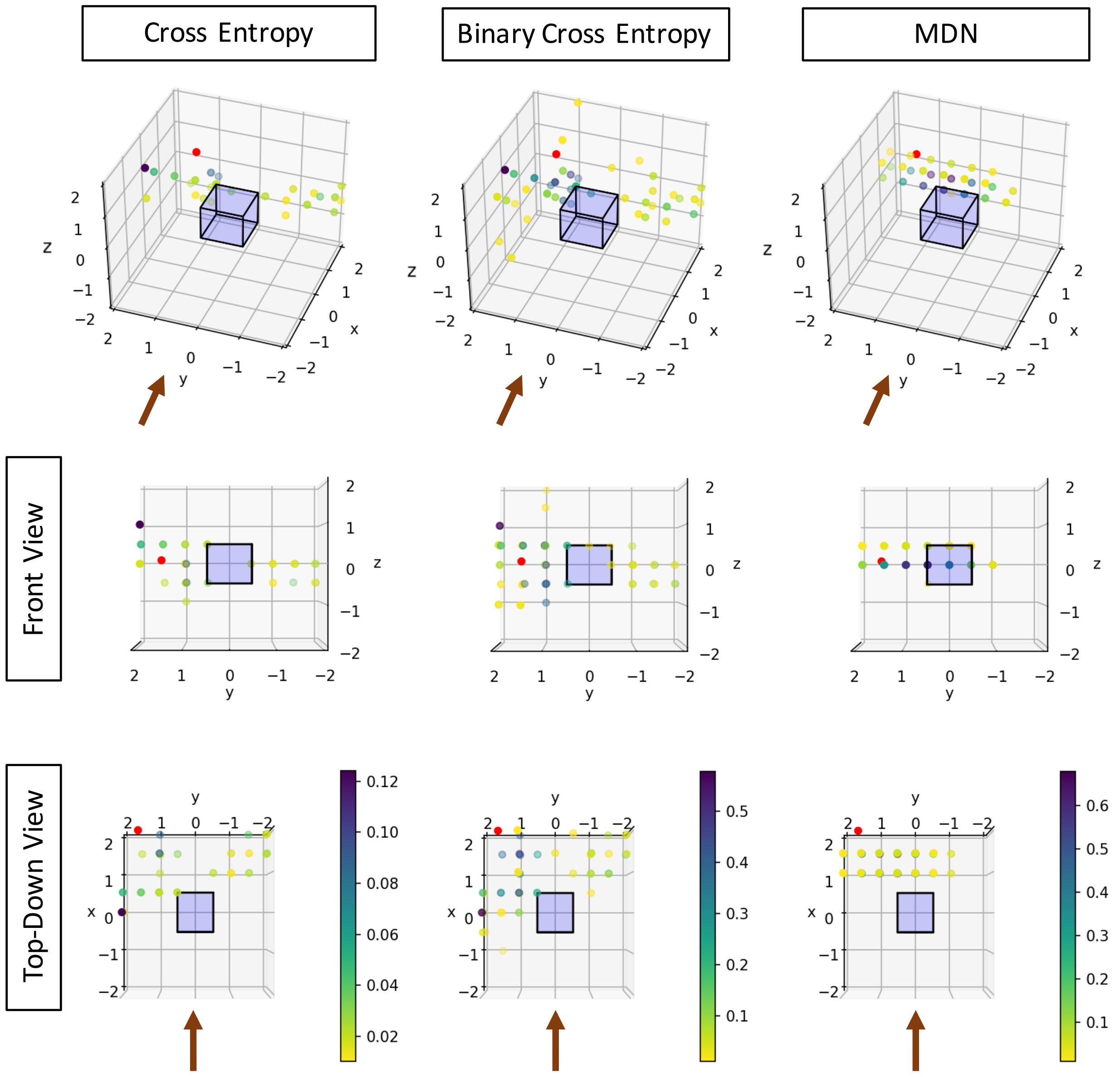


Figure 9: A comparison of the predictions obtained from a network that was trained on the cross-entropy loss (left column), on the binary cross-entropy loss (middle column) and by maximising the target probability for a Mixture Density Network (right column). The second and third row show the front and top-down view of the same scene as row 1. The view point and view direction of the simulated speaker is marked by the brown arrow. The intended position of the target is marked by the red dot (shifted by 0.1 in all directions for the purpose of better visualisation). The unseen position description is "To the left of the cube further back on the same height as the center".

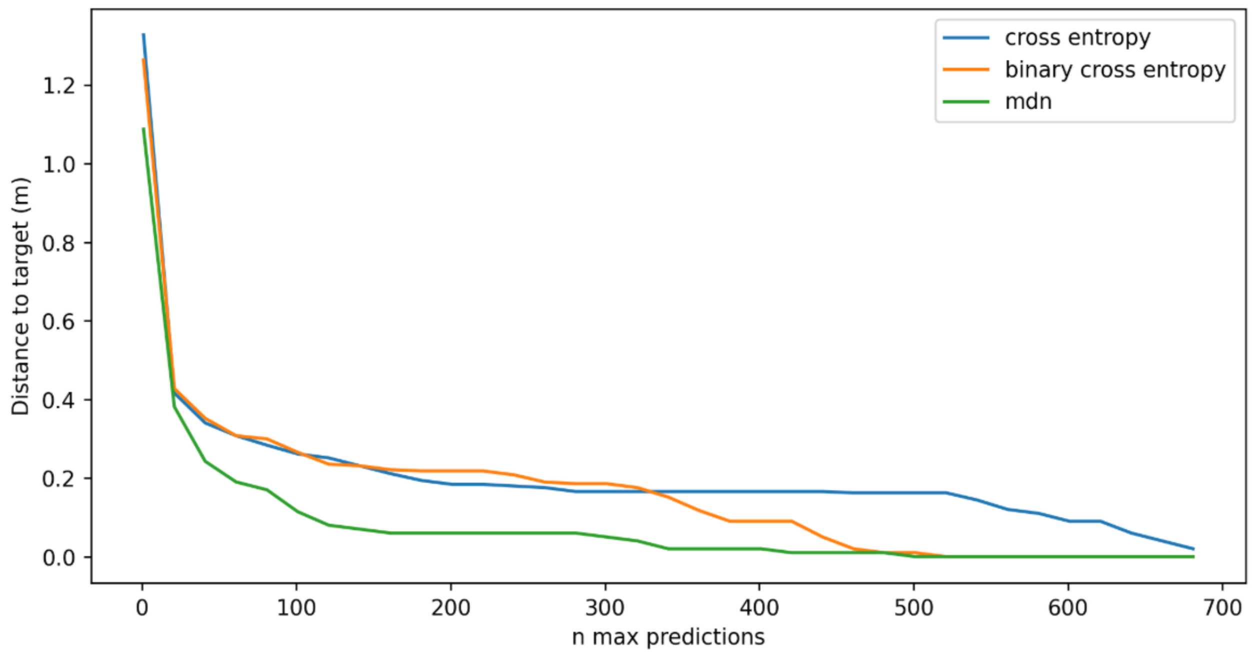


Figure 10: Comparison of the minimum distance of the top n predicted nodes to the target as a function of n for models trained on cross-entropy and binary cross-entropy and for a mixture density network. All distances plotted are the averages of the distances obtained by applying the models on 50 validation examples.

across all nodes are not normalised (as they are for the cross-entropy model or the mixture density network which outputs a normalised probability distribution). Therefore in theory the model trained on binary cross-entropy can predict every single node in the grid of being the target with probability 1 which leads to a definite relative probability of 1 at the target and explains the relatively high probability at the target which we observe.

Finally we can analyse how the distance of the predicted target to the true target change as we consider not just the maximum prediction but the maximum $n$ predictions. Here we notice that when a maximum of 20 predictions are considered the average distance to the target is almost identical for the three models. However, as more predictions are considered the average distance for mixture density network reduces a lot faster. This emphasises that while both models trained on cross-entropy and binary cross entropy have high probability predictions close to the target, they do not cover the space around the target as closely as the mixture density network does.

### 5.1.2 Architecture Ablation Study

In terms of the architecture of our network we focused on two key principles: how single words are embedded and how individual word embeddings are combined. For embedding single

| | Average Dist of max prediction from target(Training) | Average Dist of max prediction from target (Validation) | Relative probability at target (Training) | Relative probability at target (Validation) |
|---|---|---|---|---|
| Cross Entropy | 0.5281 | 1.3280 | 0.8338 | 0.1344 |
| Binary Cross Entropy | 0.5214 | 1.2637 | 0.9623 | 0.2603 |
| Mixture Density Network | 0.8420 | **1.0875** | 0.4208 | 0.1755 |

Table 1: Quantitative comparison of the networks trained on cross-entropy and binary cross-entropy and a mixture density network in the terms of the key metrics identified in Section 4.3.

words we compared learning word embeddings from scratch versus using using a pre-trained word embedder. When combining the sequences of words we investigated simply taking the average of individual embeddings and obtaining a joint representation by sequentially feeding the words into a LSTM-cell.

The 4 different models we obtain are compared in terms of the key metrics detailed above in Table 2. Focusing on the validation data we notice that there is a huge improvement from those models that learn word embeddings from scratch compared to those that use pre-trained ones. This was in fact expected as it is very difficult to learn meaningful word embeddings from only 200 training examples with roughly 15 words each. Nevertheless it is good to verify that for our problem which is highly limited in the vocabulary it uses, and may therefore learn very different word embeddings to a word embedder trained on a huge corpus of text, it is important to use a pre-trained word embedder.
Interestingly we also note an improvement in terms of the distance from the maximum prediction to the target when using a LSTM-cell to combine individual word embeddings as opposed to simply normalising the embeddings and taking the average. We did not expect this in advance as we assumed that for our position descriptions which follow very simple sentence structures and do not have any negations the position of the word in the sentence was irrelevant. In hindsight the reason why we assume combining word embeddings with an LSTM works slightly better is that we noticed that for some examples in our training data the very first attributes in a description are often the most relevant ones for the position. Following attributes are likely less important as humans automatically put the information they deem most descriptive of the position at the start.

Similarly to before we can compare the different model architectures in terms of the minimum distance to the target when considering the top $n$ predictions (see Figure 11). Doing this we find again that both models using pre-trained embeddings strongly outperform

| | Average Dist of max prediction from target (m) (Training) | Average Dist of max prediction from target (Validation) | Relative probability at target (Training) | Relative probability at target (Validation) |
|---|---|---|---|---|
| Learn embedding + average | 1.4495 | 1.8094 | 0.3673 | 0.1561 |
| Learn embedding + LSTM | 1.3439 | 1.8569 | 0.4056 | 0.1756 |
| Fasttext embedding + normalised average | 0.8420 | 1.0875 | 0.4208 | 0.1755 |
| Fasttext embedding + LSTM | 0.4132 | **0.9650** | 0.6467 | 0.1531 |

Table 2: A comparison of 4 different model architectures in terms of the average distance from the maximum prediction to the target and the relative probability at the target for training and validation data.

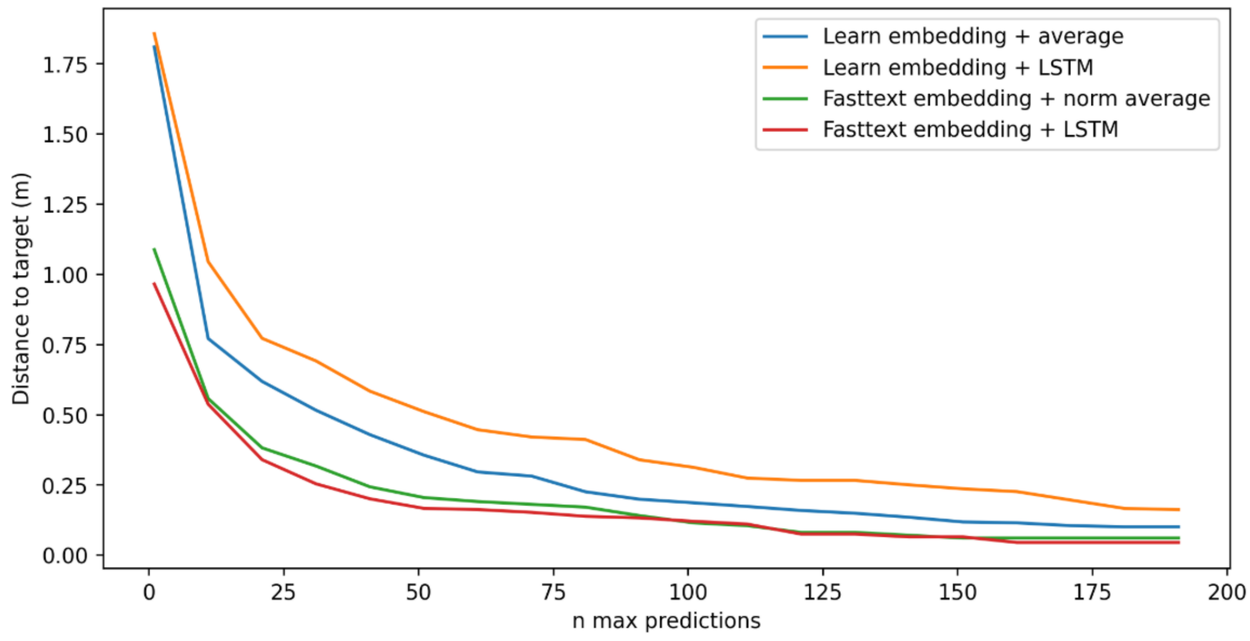


Figure 11: The minimum distance of the top $n$ predictions on unseen validation data for different network architectures.

those that do not. We also see that when learning the embeddings from scratch one obtains better results by simply combining individual embeddings by averaging them rather than feeding them through an LSTM. This is probably because it is difficult for the network to learn embeddings and LSTM weights simultaneously when training only on a very limited number of examples.

### 5.1.3 Query combination

While we have shown above that by choosing the right model architecture and the right loss functions one can make good position predictions on single descriptions, it is important to realise that in order to build a robust system one has to combine predictions from multiple descriptions. Partially this is because our model which was trained on a very limited number of examples is still inaccurate in its predictions. However, more importantly one notices again and again the inherent ambiguity in trying to describe positions with spoken languages.

Therefore in order to achieve a higher precision in the predicted poses we combine the predictions for multiple descriptions of the same position. We do so by normalising the predictions such that the maximum predicted node has a probability of 1 and then simply add the values on the global grid.

Figure 12 visualises this process for 3 descriptions of the same target pose (shown in black). Note that the viewpoint for all descriptions is the same as the one in Figure 9. In the first row we can see the contribution of the sentence "Above the chess board Table and slightly in front" and in the second row the current total distribution (which are identical for the first description). We note that while the predicted positions are indeed in front of the referenced object (red) they cover a range of vertical positions (some of which are in fact not "above" the table). The current maximum prediction is shown in red and is 97 cm away from the target position. We aim to refine the predicted position by providing another description, this time referencing a different object. The third row shows the prediction we obtain from this and we note that the maximum prediction of just this contribution is in fact very close to the target. However, when adding the normalised contribution to the total distribution we note that the maximum prediction still remains unchanged. We therefore need to specify a further description (row 5) and see that this changes the maximum total prediction to a position closer to the target, 46 cm away (row 6). This example illustrates how we can sequentially combine multiple position descriptions to obtain more accurate position predictions.

In order to quantitatively evaluate our approach we create 6 different scenarios for which we each describe a randomly chosen position with 5 different descriptions. Sometimes these descriptions reference different objects, at other times they simply rephrase a description for the same object. As before we sequentially combine the probability contributions for each of

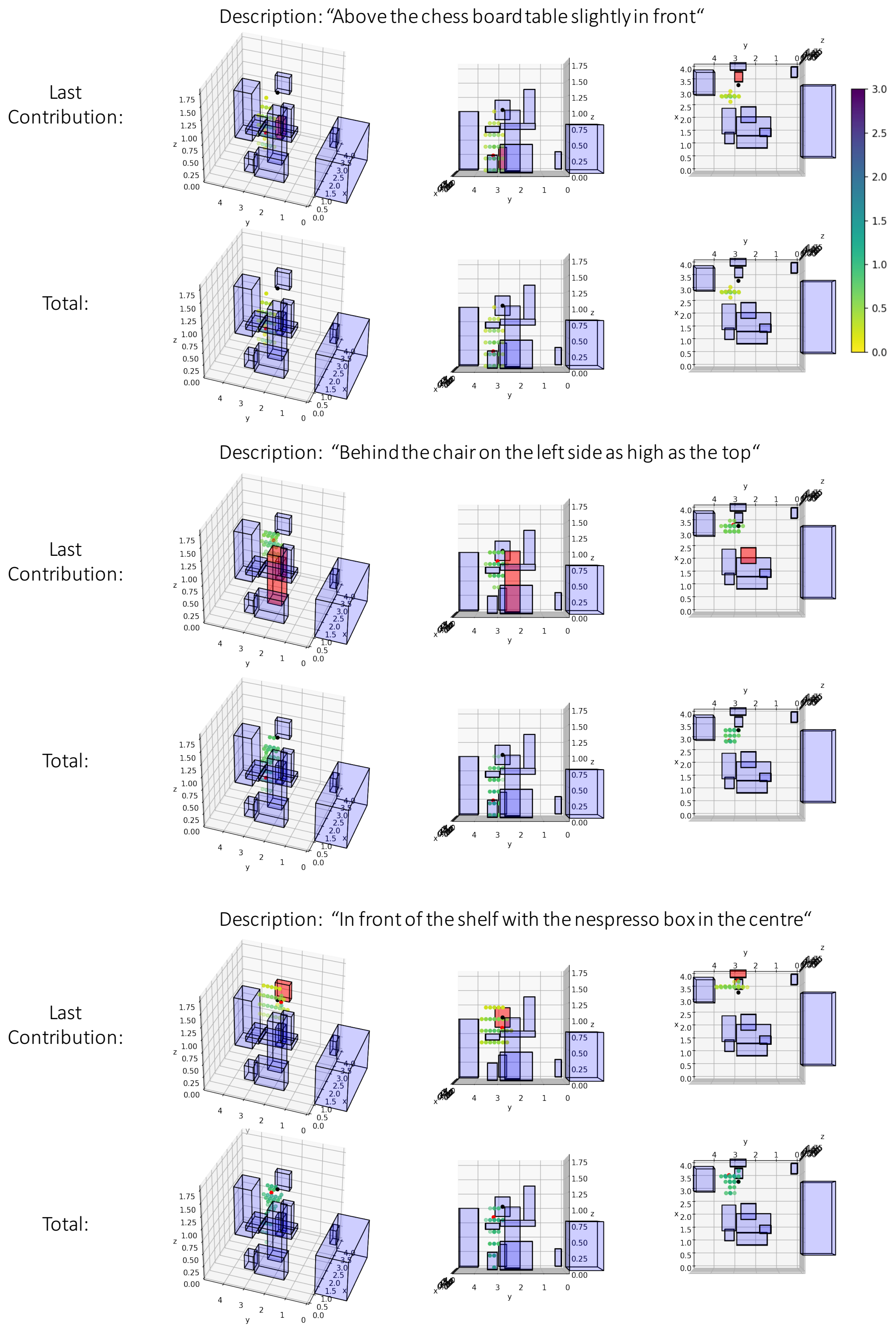


Figure 12: Visualisation of the proposed query combination. Each row shows the same probability distributions from 3 different view points. The intended target position is marked in black and the current maximum probability prediction is marked in red. From the top down we visualise the evolution of the global probability distribution (rows 2,4 and 6) after sequentially adding further position descriptions. Rows 1,3 and 5 show the probability contribution from the latest description. The object that is referenced in the descriptions is shown in red in the individual contribution.

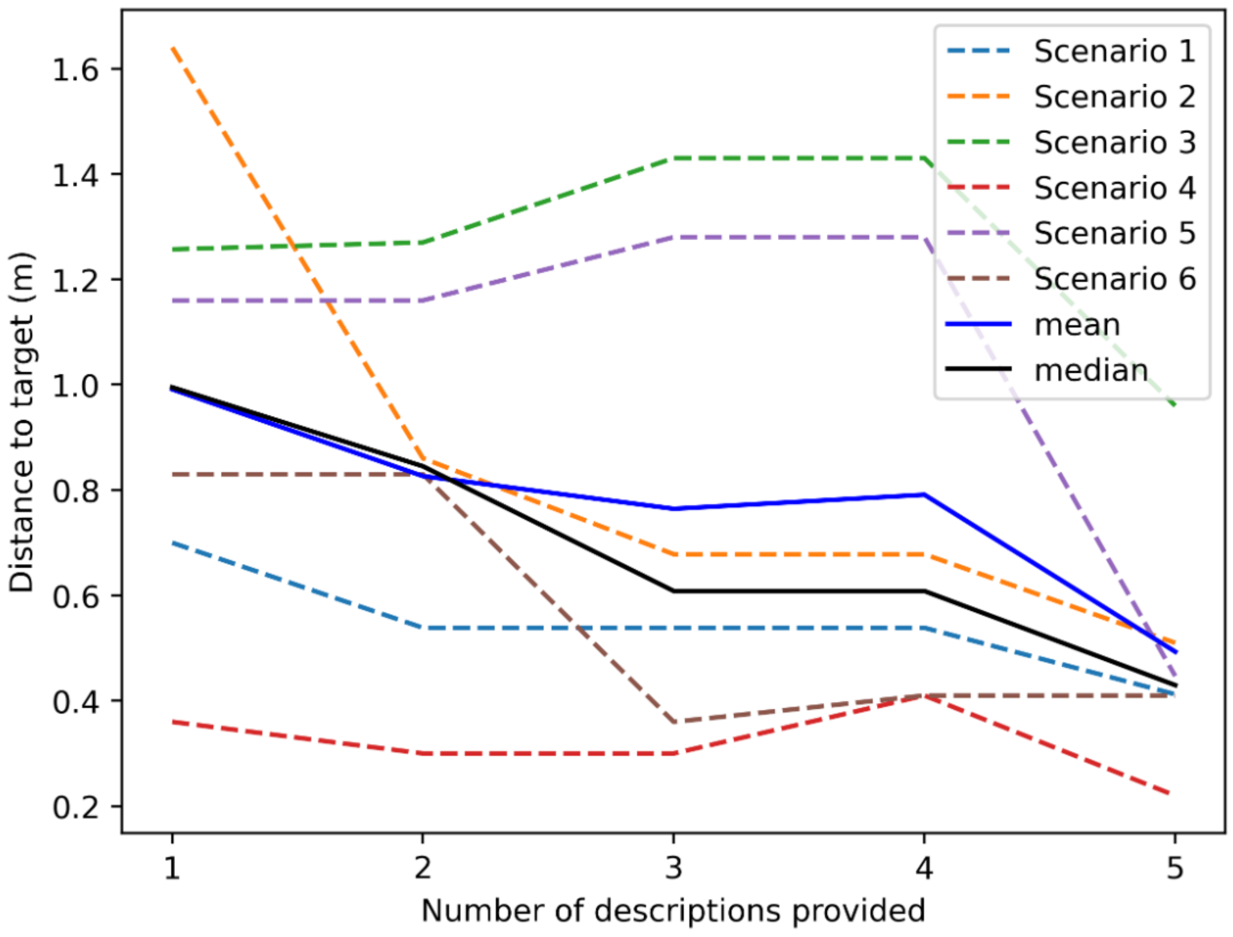


Figure 13: Quantitative Results for Query Combination: We visualise the evolution of the distance between the predicted target position and the actual target position as an increasing number of target descriptions is provided.

the descriptions in the scenario and monitor the distance between the predicted target and the actual target (see Figure 13). Here we can observe that as we increase the number of description on average the distance between the target position and the predicted position is reduced from 1 m to 0.5 m. This is in fact a very good result considering that all of the individual descriptions by itself are still very ambiguous.

## 5.2 Simulation

We test our system in a simulation of the room which we have captured using the Matterport 3D scanner. For each of the the 6 scenarios described in Section 5.1.3 we randomly sample an allowed starting pose outside of the present objects. In real-time we then render an image corresponding to this pose in Blender which we use as input for our localisation network. At the same time we predict the current target position from the speech input we provide live. We then query the graph of the environment for the shortest path between the node closest to the estimated current pose and the node closest to the current target pose[7]. The shortest path we receive is a sequence of nodes which are connected by directed edges labelled by the action required to move from the starting node to the target node. We now simply perform the action which corresponds to the first edge in the shortest path and disregard all others. We repeat this process for each time step until our predicted position and the target position correspond to the same node in the graph. At this point we add a further target description which updates the total probability distribution of the target and may lead to a new predicted target position. In the case of a new target we navigate to it as described above. Otherwise we simply provide further target descriptions, up to a maximum number of 5. After each description we record the ground truth distance of the simulated drone to the true target position.

We can see a visualisation of the trajectories for one of the scenarios in Figure 14. It shows the trajectories of the drone after it has received the first (top), the first two (middle) and the first three target position descriptions. Note that from the final positions of the trajectories we can see that the drone is closer to the target after the second position description. This is because the new contribution (Figure 15, row 2) has moved the overall target position closer to the actual target. We can also see this from Figure 13 as for scenario 4 the difference between the predicted target and the actual target decreases as we add the second description. In comparison after 3 descriptions the drone finishes at the same position as where it finished after two (left of the pink cubes in the bottom image of Figure 14). This is because adding a third description has not changed the overall predicted target as the new probability contribution only have negligible values around the actual target. Examining this latest contribution closer (Figure 15) we can see that while the prediction accurately reflects the description "higher up" it does not reflect the preposition "behind". The reason why the drone moved at all even though the target position has not change is a small imprecision during the localisation for which it is then able to correct for by returning to the starting point.

In Figure 16 we visualise the evolution of the distance between the simulated drone position and the actual target position for each of the 6 scenarios. Note that the distances observed correspond very closely to those of the predicted targets to the actual targets (Figure 13). This shows that in the simulation the localisation procedure we use works reliably. In order to make our simulation more realistic we add random noise to our predicted poses. This

[7]We convert the predicted target position to a target pose with which we can query the graph by simply specifying our target orientation to face in the positive x-direction.

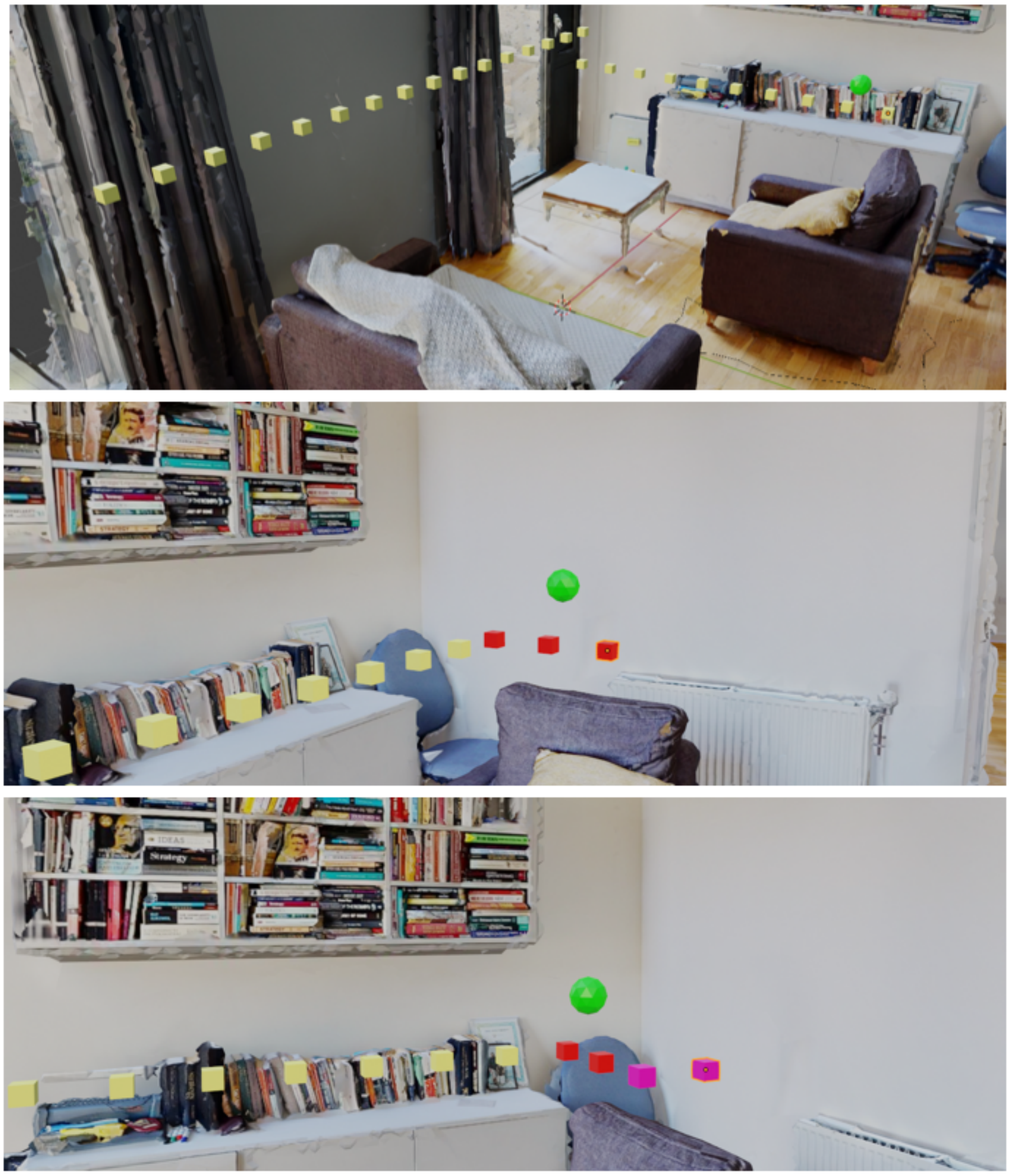


Figure 14: Visualisation of the trajectories obtained for scenario 4. The first image from the top shows the trajectory after the first description of the target position (yellow cubes). The starting point of the drone is at the center of the left most cube and the target is visualised by the green sphere. The image in the middle shows the trajectory of the drone after the second description was provided (red) and the image at the bottom shows the trajectory after the third description (pink).

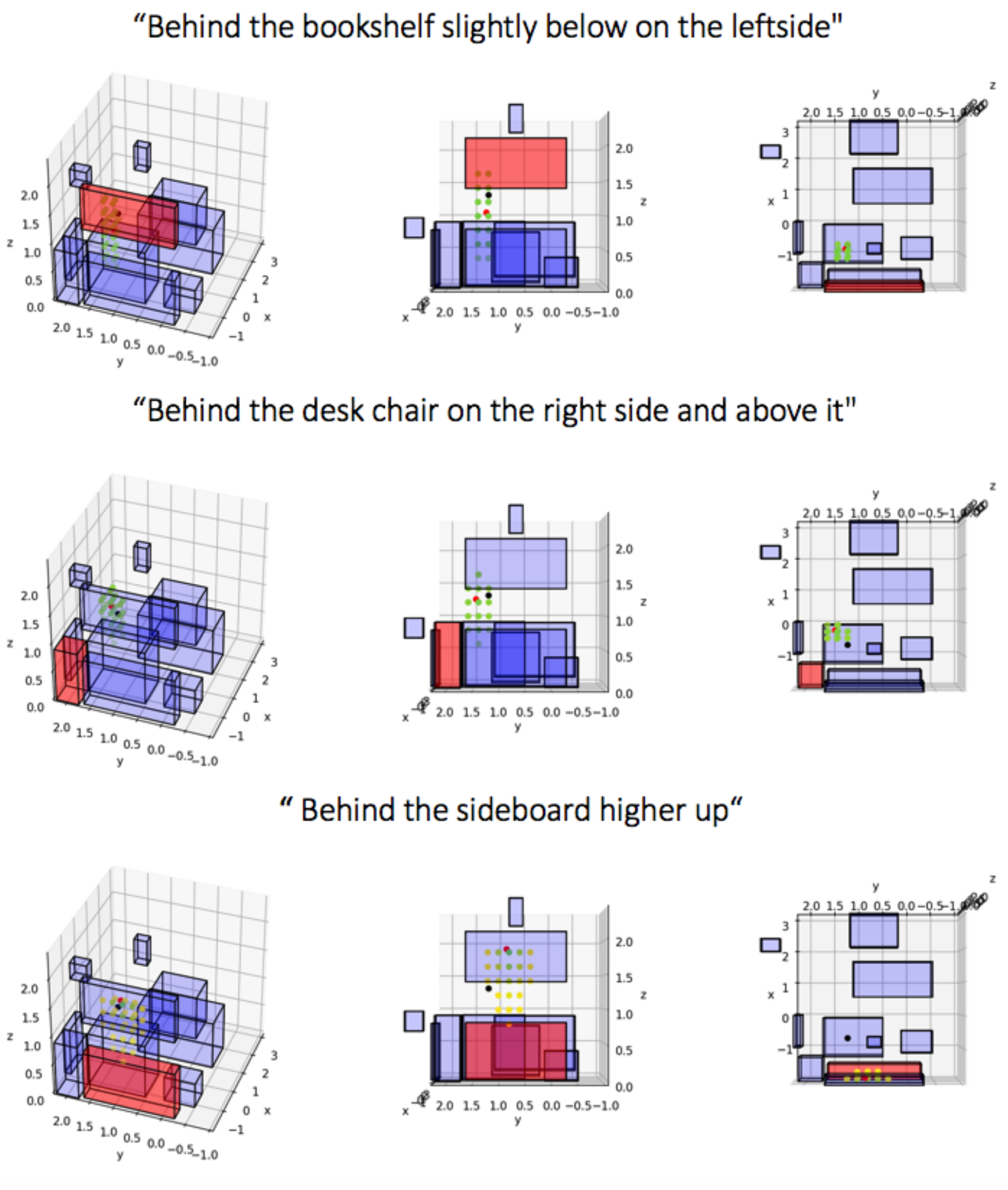


Figure 15: Visualisation of the individual probability contributions of the first 3 descriptions of scenario 4

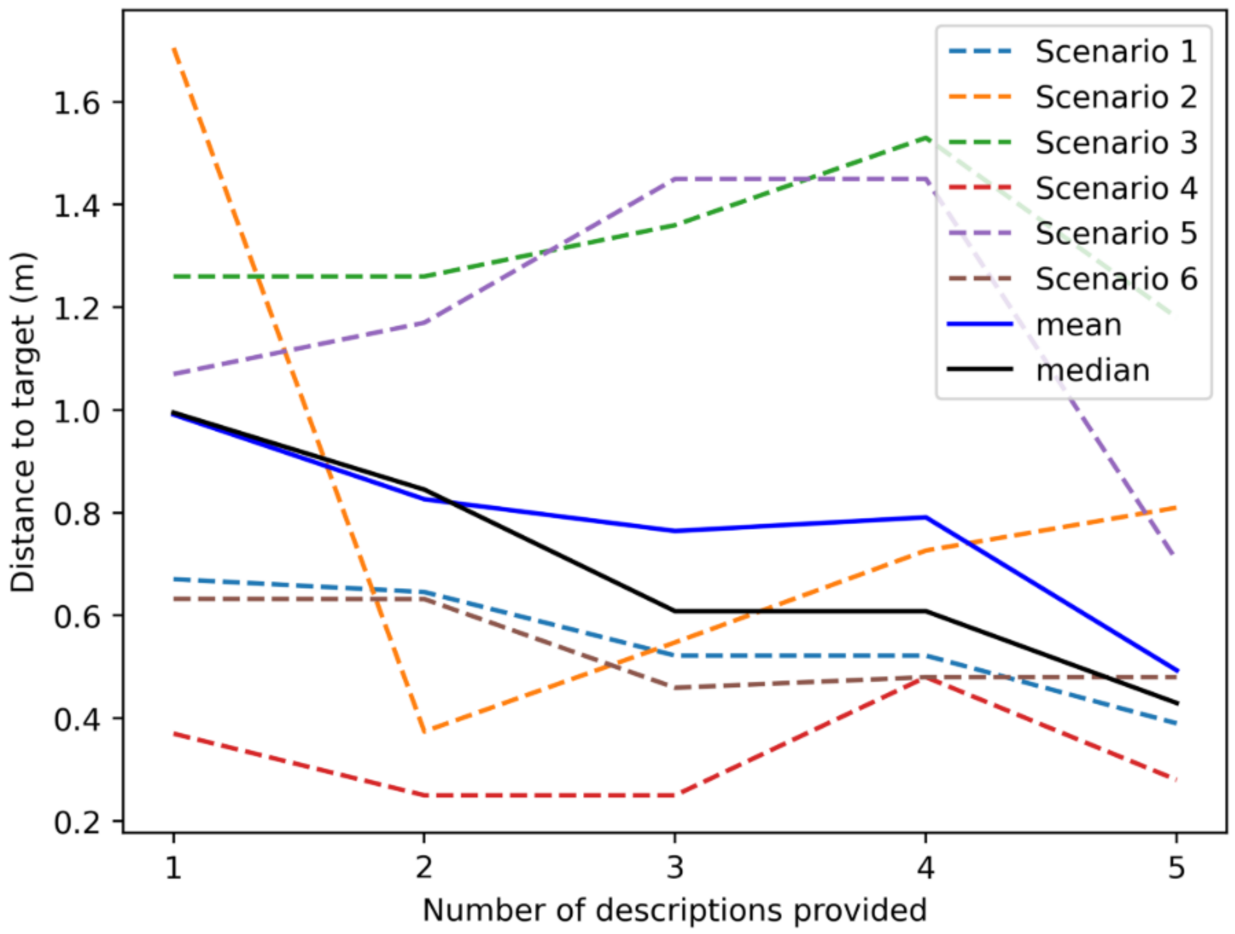


Figure 16: Simulation Results: We show the distance between the simulated final position of the drone and the target position after a varying number of position descriptions were provided.

| | noise free | position $x, y, z \pm$ unif$(0, 0.2)$ | orientation $\theta \pm$ unif$(0, 30°)$ | combined |
|---|---|---|---|---|
| Distance to the target (m) | 0.49 | 0.548 | 0.59 | 0.56 |

Table 3: Simulation results with simulated noise in the pose prediction. The distances reported are the averages over all six scenarios after all 5 target descriptions have been provided.

simulates the less precise pose predictions we get in the real world when the drone images differ slightly from the training images due to varying lighting conditions and other external factors. Table 3 shows the results we obtain when adding different levels of noise to the pose prediction. We can see that adding noise of the order of 20 cm to the position and 30° to the orientation does not affect the final distances we observe much which is promising for applying our system in the real world.

## 5.3 Real World

After extensively testing our system in simulations we evaluate it in the real world using the DJI Ryze Tello drone. The drone contains a single RGB camera in the front as well as depth sensors at the bottom and a bottom facing camera which it uses for flight stability and for holding a position. We connect to the drone via WiFi for periodically sending the next action to take. All speech recording runs through the computer as the drone is not equipped with a microphone. Just as we did for the simulation we evaluate the performance of the system by testing it on a series of test scenarios. Table 4 shows the final differences in position we estimated for each of the 5 scenarios. The final distances we observe range from 0.5 m to 0.9 m with an average distance of 0.62 m. These distances are similar to the results we obtained in the simulation which is very positive as it shows that we can effectively train our system on simulated data and still apply it in the real world. Figure 17 and 18 show images that were taken from videos of the drone performing the first and the second scenario.

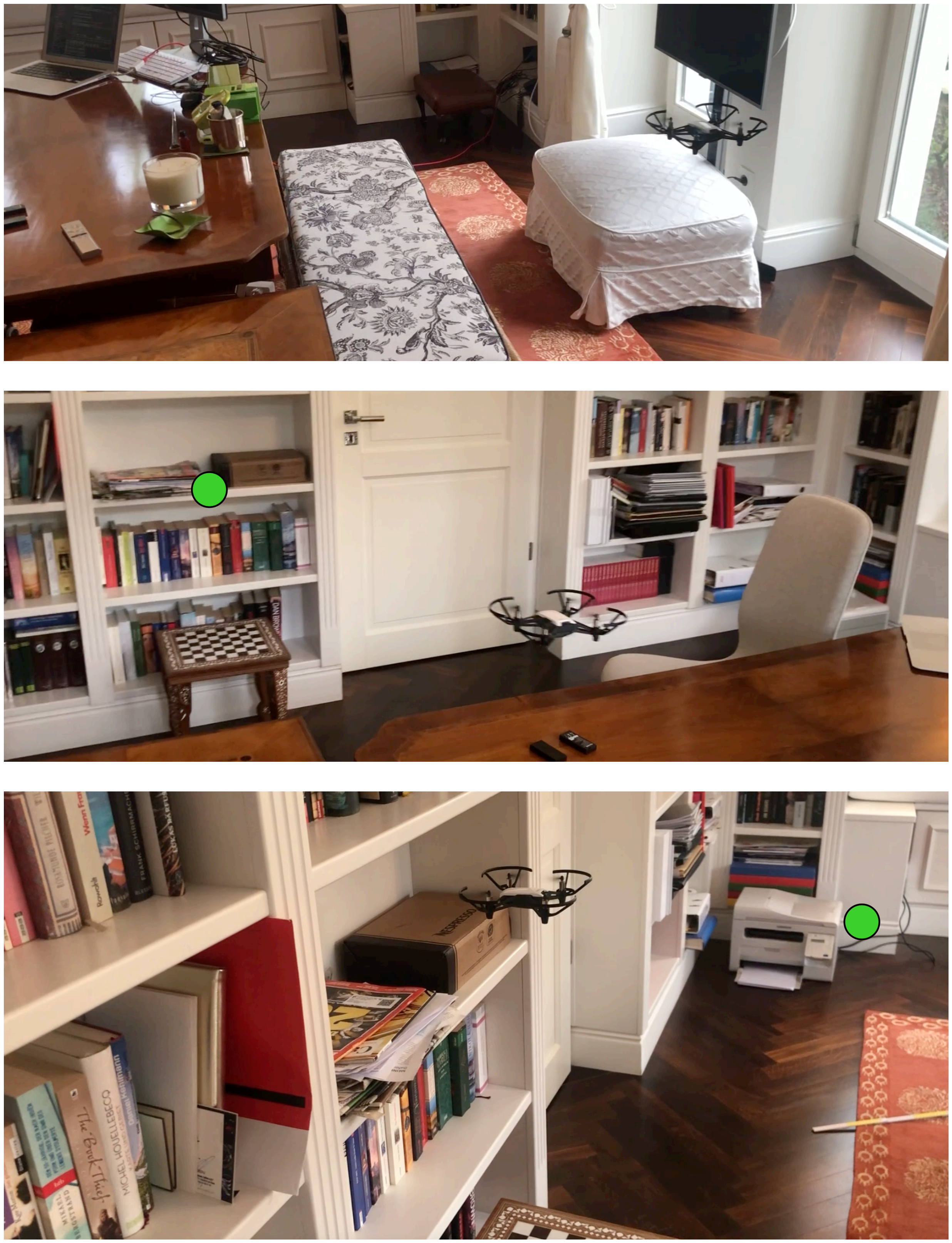


Figure 17: Visualisation real world scenario 1: The images taken are screenshots from a video filming the drone while following the commands of the first scenario in the real world. The images correspond to the position at the start (top), midway through flight (middle) and the final position (bottom) of the drone. The intended target is visualised by the green dot. See Table 4 for the position descriptions.

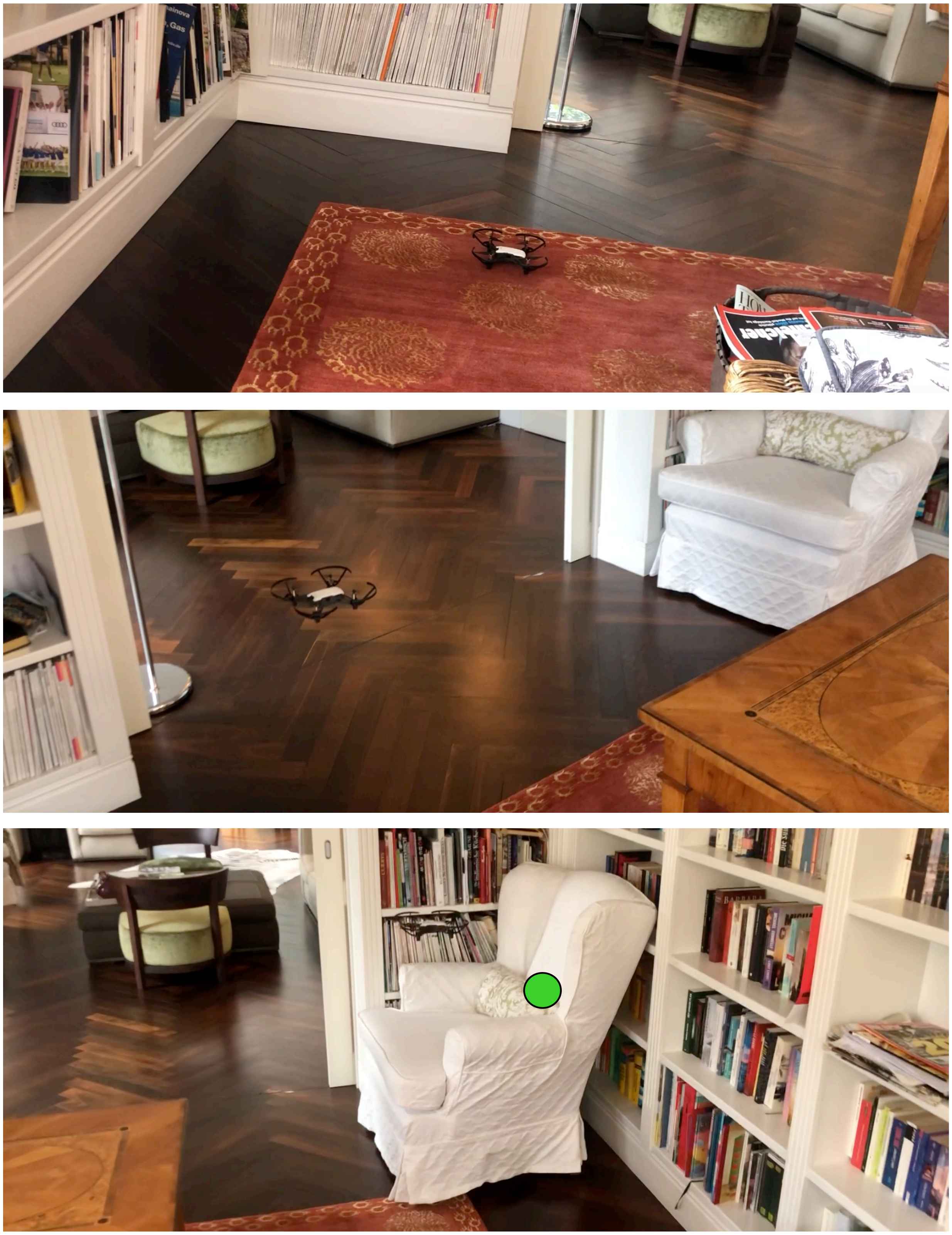


Figure 18: Visualisation real world scenario 2: The bottom image shows the final position of the drone. The target position is marked by the green dot. See Table 4 for the position descriptions.

| Scenario | Descriptions | Estimated final distance to target (m) |
|---|---|---|
| 1 | "Above the chess board table slightly in front"<br>"Behind the chair on the left side as high as the top"<br>"In front of the shelf with the Nespresso box[8] in the center" | 0.5 |
| 2 | "To the right of the armchair on central height"<br>"To the left of the chess board table" | 0.9 |
| 3 | "Underneath the side table"<br>"Behind the bench but on the left on central height"<br>"Behind the box with newspapers" | 0.5 |
| 4 | "Above the printer"<br>"To the left of the couch in the back on central height" | 0.6 |
| 5 | "To the right of the computer screen on central height"<br>"To the right of the table higher up"<br>"To the left of the couch on central height" | 0.6 |

Table 4: Results for real world scenarios: The distances quoted are the estimated final distances of the drone after it was provided with all the position descriptions.

## 6 Conclusion and Future Work

In this project we developed a system that uses human descriptions of positions to navigate a drone to the desired position. In order to do so we made a first attempt at learning spatial representations of spoken position descriptions which can be applied more generally to indoor robotics. The first step in our procedure was to create a dataset containing a position around a reference object and the semantic description of that position. We used this dataset to compare and contrast the results we obtained for different model architectures and loss functions. We demonstrated that using a pre-trained word embedding is crucial, while preserving information about the order of the words in the description by using an LSTM can give an extra increase in accuracy. Furthermore we show that directly predicting position probabilities is sub-optimal and one can achieve smoother, more accurate and more meaningful prediction by predicting parameters of a Gaussian Mixture Model. We show that by combining predictions made from individual position descriptions the combined distribution we obtain is more accurate in estimating an often ambiguous position and resilient to individual miss-predictions.

Having evaluated the system in isolation we proceeded to test it in simulated scenarios. Here we directly used the predictions made by the model to navigate a simulated drone to the predicted target positions and evaluate the performance. When specifying up to 5 descriptions

[8]Note that "shelf with the Nespresso box" was treated as a single object.

of the target position we achieve an average accuracy in the final position of 0.5 m. Finally we have tested our approach on a small commercial drone and show that our system can be successfully transferred to the real world.
The achievements we have made so far can be extended in numerous ways:

- **Viewpoint Dependency:** An important extension of the current system for deployment to real robots interacting with humans is to enable a view point dependency for all position descriptions. This viewpoint should be either the position of the speaker (which has to be estimated) or the position of the robot itself, depending on preference. This viewpoint dependency can be easily achieved by simply collecting more training data from random speaker positions and using this position as an input to the fully connected layers of the network alongside the sentence encoding.

- **Ambiguity:** Secondly, we would like to explore the ambiguity of many position descriptions further. Prepositions such as "next to" or "nearby" can be used to describe many potential target positions. While preliminary experiments have shown that due to its multi-modality our model can learn these ambiguous descriptions too, we would like to investigate this in more detail.

- **Multiple Reference Objects:** Additionally we are planning on training our model on multiple reference objects to allow handling of a wider class of prepositions, e.g. "between" and "in the middle of".

- **Richer Model:** Finally, we would like to explore more complex and powerful network architectures to improve the accuracy in our predictions.

# References

[1] Daniel Haun, C. Rapold, G. Janzen, and S. Levinson. Plasticity of human spatial cognition: spatial language and cognition covary across cultures. *Cognition*, 119(1):70–80, April 2011.

[2] Alexander Kranjec, Gary Lupyan, and Anjan Chatterjee. Categorical biases in perceiving spatial relations. *PLOS ONE*, 9(5):1–9, 05 2014.

[3] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[4] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. From coarse to fine: Robust hierarchical localization at large scale, 2018.

[5] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Convolutional networks for real-time 6-dof camera relocalization. *CoRR*, abs/1505.07427, 2015.

[6] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning, 2017.

[7] Eric Brachmann and Carsten Rother. Learning less is more - 6d camera localization via 3d surface regression, 2017.

[8] Parisa Kordjamshidi, Paolo Frasconi, Martijn Van Otterlo, Marie-Francine Moens, and Luc De Raedt. Relational learning for spatial relation extraction from natural language. In *Proceedings of the 21st International Conference on Inductive Logic Programming*, ILP'11, page 204–220, Berlin, Heidelberg, 2011. Springer-Verlag.

[9] Robert Ross John A. Bateman, Joana Hois and Thora Tenbrink. A linguistic ontology of space for natural language processing. *Artificial Intelligence*, 174:1027 – 1071, 2010.

[10] Masoud Rouhizadeh, Richard Sproat, and Bob Coyne. Collecting spatial information for locations in text-to-scene systems. 04 2012.

[11] Angel Chang, Manolis Savva, and Christopher D. Manning. Learning spatial knowledge for text to 3D scene generation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2028–2038, Doha, Qatar, October 2014. Association for Computational Linguistics.

[12] Kaveh Hassani and Won-Sook Lee. Visualizing natural language descriptions: A survey. *CoRR*, abs/1607.00623, 2016.

[13] Tiago Ramalho, Tomás Kociský, Frederic Besse, S. M. Ali Eslami, Gábor Melis, Fabio Viola, Phil Blunsom, and Karl Moritz Hermann. Encoding spatial relations from natural language. *ArXiv*, abs/1807.01670, 2018.

[14] Bob Coyne and Richard Sproat. Wordseye: An automatic text-to-scene conversion system. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 487–496, New York, NY, USA, 2001. Association for Computing Machinery.

[15] Christian Spika, Katharina Schwarz, Holger Dammertz, and Hendrik P. A. Lensch. AVDT - Automatic Visualization of Descriptive Texts. In Peter Eisert, Joachim Hornegger, and Konrad Polthier, editors, *Vision, Modeling, and Visualization (2011)*. The Eurographics Association, 2011.

[16] T. Winograd. Procedures as a representation for data in a computer program for understanding natural language. 1971.

[17] Weituo Hao, Chunyuan Li, Xiujun Li, Lawrence Carin, and Jianfeng Gao. Towards learning a generic agent for vision-and-language navigation via pre-training, 2020.

[18] Microsoft Azure Cognitive Services. Microsoft speech sdk for python (1.13.0), 2018.

[19] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[21] C. Bishop. Mixture density networks. 1994.

[22] E.W. DIJKSTRA. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.