Real-time Opinion Aggregation Methods for Crowd Robotics

Elliot Salisbury, Sebastian Stein and Sarvapali D. Ramchurn
Agents, Interaction and Complexity Research Group
School of Electronics and Computer Science
University of Southampton, UK
{e.salisbury,ss2,sdr}@ecs.soton.ac.uk

ABSTRACT

Unmanned Aerial Vehicles (UAVs) are increasingly becoming instrumental to many commercial applications, such as transportation and maintenance. However, these applications require flexibility, understanding of natural language, and comprehension of video streams that cannot currently be automated and instead require the intelligence of a skilled human pilot. While having one pilot individually supervising a UAV is not scalable, the machine intelligence, especially vision, required to operate a UAV is still inadequate. Hence, in this paper, we consider the use of crowd robotics to harness a real-time crowd to orientate a UAV in an unknown environment. In particular, we present two novel real-time crowd input aggregation methods. To evaluate these methods, we develop a new testbed for crowd robotics, called CrowdDrone, that allows us to evaluate crowd robotic systems in a variety of scenarios. Using this platform, we benchmark our real-time aggregation methods with crowds hired from Amazon Mechanical Turk and show that our techniques outperform the current state-of-the-art aggregation methods, enabling a robotic agent to travel faster across a fixed distance, and with more precision. Furthermore, our aggregation methods are shown to be significantly more effective in dynamic scenarios.

Categories and Subject Descriptors

I.2.9 [**Robotics**]: Operator interfaces; I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Human Factors, Experimentation

Keywords

real-time crowd control; real-time human computation; crowdsourcing; crowd robotics

1. INTRODUCTION

Unmanned aerial vehicles (UAVs) are increasingly becoming central to many commercial, humanitarian, and law enforcement applications. For example, maintenance engineers use UAVs for remote inspection, and determine whether buildings, machines, or power lines require repair [7], online distributors[1] and humanitarian organisations [3] are using UAVs to transport small packages, and law enforcement agencies are using UAVs to monitor traffic and assess hazardous situations [8]. Most of these applications currently require a skilled pilot, and cannot be automated. Engineers maintain many different structures, that can be damaged in many different ways, thus they require accurate computer vision to be able to identify the areas in need of repair. Drone couriers delivering aid or packages require natural language processing to navigate to a given description of the drop target. And law enforcement agencies need vision comprehension to be able to understand when people are in danger.

Consequently, given that these tasks require the intelligence and skills of a pilot, using a pilot to control UAVs for these applications has its drawbacks. Piloting a UAV is a taxing process, and a single operator will eventually suffer from fatigue and cognitive overload [2]. Given the human limitations of operators, they may be unable to both effectively operate the robot and simultaneously review all sensor streams. Furthermore, it is hard to envisage that such an approach would scale to hundreds or thousands of UAVs in the future.

Unfortunately, automating these tasks requires artificial intelligence capable of understanding natural language, and flexible image recognition and comprehension, on a level similar to that of a human. This goal is still a long way off [5]. Crowd robotics can help bridge the gap between the limitations of artificial intelligence and the desired outcome of automated UAVs. Crowd robotics is the process of using a real-time crowd of non-experts to influence the control of a robotic agent, in this case a UAV. The crowd retains the human intelligence to be able to follow natural language instructions and comprehend camera footage, but is not limited to a single pair of eyes. Crowd robotics enables many more people to observe the camera, and in doing so, they are more likely to observe and accurately identify important details. Furthermore, while a pilot may fatigue, a crowd robotics team is a tireless workforce. More specifically, individuals of the crowd that tire and whose performance drops, can be replaced by new crowd members. As such, the pool of workers is always being replenished with fresh individuals.

A number of attempts have been made in the past to deploy crowd robotics systems. For example, online crowds control large telepresence robots to roam museum atriums [10] or to steer mobile webcams [4], but these are known and controlled environments, where fast response times and

[1]Such as Amazon Prime Air

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*
Copyright © 2015, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

efficient path planning is not critical. In these settings, aggregation methods were used to combine the inputs of the individual crowd workers and thereby harness the collective intelligence of the crowd. However, by focusing on static environments, they do not address a number of key challenges in the UAV setting. UAVs typically operate in unknown, dynamic environments. Thus, UAVs have strict time constraints to react to changes in the environment in real time. Furthermore, UAVs have limited flight time, and should endeavour to keep moving, minimising stationary time or time spent backtracking. Additionally, these applications have commercial interests at stake. As crowd robotics relies on an open system (i.e., anyone may participate), unreliable members of the crowd may negatively impact the system and therefore should be filtered out to ensure a desirable outcome.

Solving these challenges requires real-time methods of aggregation that focus on the constraints and abilities of UAV hardware, while also limiting the influence of unreliable crowd members. These aggregation methods need to be computed *online*, so that they can be applied to robotic agents in real time. In traditional crowdsourcing, methods of opinion aggregation are often applied offline [11] and require a lot of computational power that is too prohibitive for the real-time nature of our scenarios. Furthermore, other methods of consensus have been achieved through the use of machine learning techniques, but given that these scenarios have no ground truth available, it is difficult to apply those techniques here. In addition, there is currently no principled approach to evaluating these aggregation methods and measuring their effectiveness under differing use cases and conditions.

Against this background, we first introduce a novel framework for evaluating aggregation methods employed by a crowd robotics system. Furthermore, we describe novel aggregation methods designed for the UAV domain. In more detail, this paper advances the state of the art in the following ways:

1. We present CrowdDrone, an application that allows multiple users to simultaneously influence the path taken by a UAV in real time.

2. We introduce a principled approach to measuring and comparing different real-time input aggregation methods.

3. We describe novel opinion aggregation methods for real-time control of robotic agents. These aggregation methods focus on filtering out unreliable crowd members *online* and translating from a simplified interface for crowd input, to a more expansive set of outputs for robotic control.

4. We test these aggregation methods using a real crowd gathered from Amazon Mechanical Turk. In total, 480 crowd members took part, with up to 13 simultaneous users. We show that these aggregation methods enable a robotic agent to travel faster across a fixed distance, and with more precision, than the current state-of-the-art algorithm for aggregating real-time crowd input.

The rest of this paper is structured as follows, next in Section 2 we discuss the related work. Then in Section 3 we describe CrowdDrone, a novel testbed framework for crowd robotics. Section 4 goes on to describe the model behind input aggregation, and the novel real-time aggregation methods introduced in this paper. Finally in Section 5 we evaluate these aggregation methods in two different scenarios, and present the results.

2. RELATED WORK

There are several real-world examples where web users have been given the opportunity to control robotic agents. In 1998 a telepresence robot was allowed to roam the atrium of a museum [10], and this is now becoming a more common occurrence in museums.[2,3] However, this method of control requires the environment and the location of the exhibits to be known in advance. This is not suitable for our UAV applications, where the environments are typically unknown.

More recently, [4] developed the Legion system, in which a real-time synchronous crowd can collaboratively interact with any user interface. The crowd may be asked to perform a wide range of tasks, such as word processing, data entry, or pilot a small remote-controlled mobile webcam. Legion defines a number of methods for aggregating input from the crowd, some of which attempt to correct for noisy or erroneous crowd members. Notably, the *Leader* aggregator, in which a single crowd member that most agrees with the crowd assumes sole control of the agent, was the most successful for real-time control of the mobile camera. For this reason, we will use it as a benchmark for our crowd robotics system. However, the majority of the aggregators specified in Legion are not designed for real-time robotic control applications, and are instead more general, for the sharing of any interface, and therefore do not apply to our setting.

Crowd robotics requires that we aggregate the input of many individuals in the crowd in order to create a timely and more effective operator than any non-expert individual in the crowd. To achieve more effective operations, we require a method of selecting input that is conducive to the task at hand. In this context, we recognise that a number of techniques could be used for this purpose. Namely, machine learning techniques, specifically supervised learning, are capable of learning rules that map input to favourable output, but these techniques require that we have knowledge of when an outcome is favourable [9]. However, in our applications, we do not know the environment, or the correct identification of targets (e.g., damage or dropzones), and computer vision is not currently capable of reliably verifying this [5]. Hence, machine learning techniques would not be applicable to our scenarios. Instead we turn to the concept of agreement metrics [6]. These metrics give a score of how much consensus there is amongst the input, trusting that combined human intelligence can inform us if an action is beneficial to the task at hand. In particular, voting protocols as a means of aggregating highly agreed-upon votes, is particularly useful when votes need to be aggregated in real-time, due to the low computational complexity involved [6]. However, there has been little work done to prove the viability of voting protocols in real-time aggregation settings. For a crowd robotics aggregation method to be viable, it needs to be capable of accurately representing the crowd's opinion, be calculable in real-time, and show robustness to unreliable crowd members. To date, the viability of these

[2] `http://www.afterdark.io`

[3] `http://www.nma.gov.au/engage-learn/robot-tours`

real-time voting protocols in a robotic setting has not been tested. We are the first to evaluate these protocols using a real-time crowd of workers from Amazon Mechanical Turk.

Next we discuss CrowdDrone, a system that enables us to evaluate these real-time opinion aggregation methods in the space of UAV control.

3. CROWDDRONE

A crowd robotics system must be capable of gathering and maintaining a real-time crowd, recruited online using a crowdsourcing platform, such as Amazon Mechanical Turk[4] (AMT). Furthermore, a crowd robotics system requires a robot that can maintain a constant connection to the internet, such that it can stream sensor output to the crowd, then act upon their feedback. Given that a crowd robotics system is open to anyone online, the experience, skill, and reliability of crowd members may vary greatly. For this reason crowd robotics requires opinion aggregation methods that can filter out this noise. Furthermore a crowd robotics system must also be scalable, as crowd members dynamically join and leave, the system should remain real-time such that the feedback from the crowd can influence the path of the robot in a timely manner.

To address these requirements, we developed the open-source platform, CrowdDrone, available for download online.[5] CrowdDrone is powered by standard web technologies (i.e., HTML, JavaScript, WebSockets) and a well-known robotics library, ROS[6], that enables seamless portability from real robots to simulated robots. CrowdDrone can use simulated environments to enable researchers to quickly develop dynamic environments, that would otherwise be cost prohibitive to build in reality. We use an established simulation environment, Gazebo.[7] It is developed and maintained by the Open Source Robotics Foundation, and it realistically simulates robots and environments to help avoid the common problems of robotics, such as short battery life and dangerous behaviours towards the robot or human operators.

CrowdDrone applies the crowd robotic principles mentioned above to the control of a UAV, specifically a quadcopter. A multi-rotor UAV was chosen, as it is more commonly used by drone delivery systems and law enforcement agencies. In this paper the robot is simulated in a physically realistic environment, as it enables fast creation of interesting test scenarios (see Section 5.1), but the system could just as easily be applied to a real robot.

CrowdDrone uses the retainer model [1] to hire and maintain a real-time crowd from AMT. The retainer model is a method of pre-hiring a crowd and alerting them to all participate simultaneously when the task is ready. The crowd then observes the imagery from a camera on board the robot, and votes on the direction they wish the robot to move next (see Figure 1). These votes are combined through various real-time input aggregation methods (discussed in Section 4.1).

Within crowd robotics systems, most members of the crowd are non-experts, and so we must ensure that the control scheme is simple and accessible (i.e., it is safe to assume all AMT users have keyboards, whereas a control pad is not a sensible control scheme). For this reason, we use a reduced set of actions for a UAV. Even though a UAV is capable of four or more degrees of freedom, we constrain the input set to influence only two degrees, moving forward or backward, and yawing left or right. As such, the control scheme can be reduced to simply the four arrow keys on a standard keyboard. The arrows on the keys intuitively represent the input actions.

In what follows we first model the decision problem to be solved in the CrowdDrone platform. We then go on to describe the novel voting methods for real-time aggregation of crowd member inputs.

4. INPUT AGGREGATION MODEL

As in Section 3, CrowdDrone consists of three main components. A robotic agent, a real-time crowd, and an input aggregation algorithm. In more detail, the agent, a drone, has a set of actions that it may perform, Θ. As previously mentioned, the control method is simplified, such that the agent is capable of only performing four actions, moving forward ($\uparrow$), backward ($\downarrow$), yawing left ($\leftarrow$), and yawing right ($\rightarrow$).

$$\Theta = \{\uparrow, \downarrow, \leftarrow, \rightarrow\} \tag{1}$$

The actions of the drone may be dictated by votes generated by a crowd. In more detail, we assume the crowd is comprised of N members, $U = \{1, 2, \cdots, N\}$. Each member, u, is capable of voting for actions they wish the agent to perform, $v_u^t \in \Theta$, where, $t \in \mathbb{R}_{\geq 0}$, is the timestamp of when this vote is received, and $t = 0$ is the start of the experiment. This voting process allows crowd members to express their opinion about which direction they wish the agent to go, by interacting with the given interface, such as pressing the $\uparrow$ key on their keyboard to suggest that they wish the agent to move forward.

Let $V_u = \left\{v_u^{t_1}, v_u^{t_2}, \cdots, v_u^{t_k}\right\}$ denote the set of all actions performed by the member u, and V is the union of these over all members of the crowd, that is:

$$V = \bigcup_{u \in U} V_u \tag{2}$$

We then define an input aggregation function that combines the votes of the members, into a single stream of output actions that the agent will perform.

$$f : 2^U \times 2^V \times \mathbb{R}_{\geq 0} \longrightarrow \Theta \tag{3}$$

The function f takes as parameters, U the set of all members; V the set of all votes; t the current time; and outputs an action $\theta \in \Theta$.

If all the inputs of the crowd were known in advance, function f could be computed offline with traditional crowdsourcing methodologies. However, due to the real-time nature of our applications, f has to be computed online, with real-time crowdsourcing. This means that the output θ at any time t can only be computed based on all inputs received prior to time t (rather than all those beyond t in the offline case). Hence, in the next section we explore the algorithms that implement f in real-time.

4.1 Input Aggregators

Aggregating votes from a real-time source presents a number of key challenges:

[4] http://www.mturk.com
[5] https://github.com/ElliotSalisbury/CrowdDrone
[6] http://www.ros.org
[7] http://www.gazebosim.org

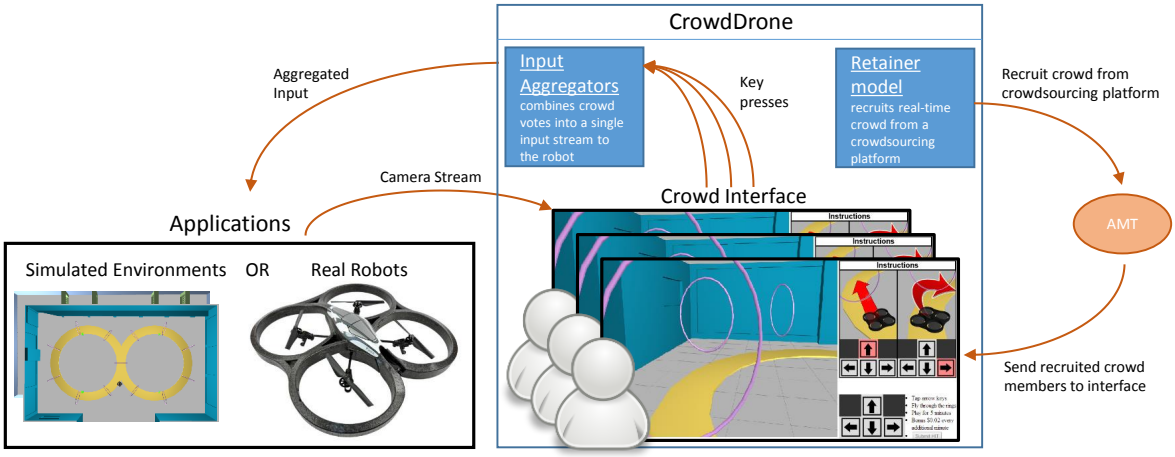


Figure 1: The CrowdDrone System Diagram

- **Computational Complexity:** as the crowd size grows large, so too will the rate of input received per second. The input aggregation algorithms must remain responsive even at large scale.
- **Accuracy:** due to unreliable or malicious crowd members, we need to be able to filter out their input, and aggregate input beneficial to the task at hand. To accurately represent the crowd, we want to perform the actions that move the robot closer to where the majority of the crowd wishes to go.

The following methods have been chosen such that they can be applied in real time, in unknown environments without a ground truth. Thus, in what follows, we describe the voting strategies used for the input aggregation evaluated as part of the CrowdDrone system. We then go on to compare the various aggregation methods (Section 4.2), discussing their relative strengths and weaknesses.

4.1.1 Mob

The *Mob* aggregator as implemented in Legion [4], is the simplest and most naive approach to combining the input of the crowd. Whenever a member in the crowd submits a vote, the agent will immediately perform that action, until a new vote is received. See [4] for implementation details.

4.1.2 Leader

The *Leader* aggregator was originally described in [4]. The aggregator hosts elections of duration d seconds, selecting the crowd member who has agreed most often with the other crowd members' votes and then allows that member full control of the robotic agent for the duration of the next election. See [4] for implementation details.

4.1.3 Real-Time Majority

The *Real-Time Majority* aggregator calculates a winner as follows. In brief, whenever a member in the crowd submits a vote, the *Real-Time Majority* aggregator gathers all the votes over the past d seconds and calculates the action θ that most crowd members voted for, while weighting the influence of those users based on their past agreement with the crowd.

In more detail, the *Real-Time Majority* aggregator first requires a weight for each crowd member u to reduce the influence of unreliable crowd members. This weight is iteratively recalculated every d seconds and is a value proportional to that member's agreement with the crowd. To calculate this weighting, it is first beneficial to define a function (Equation 4) that retrieves the latest vote from a member, and a function (Equation 5) that returns the set of members whose latest vote was for a given action θ in the past d seconds.

$$latestVote(u) = v_u^{t'} \mid \forall v_u^{t''} \in V : t' \geq t'' \tag{4}$$

$$\begin{aligned} votedFor(t,\theta) = \{u \mid \exists v_u^{t'} &= latestVote(u) \ \wedge \\ v_u^{t'} &= \theta \quad \wedge \\ t - d < t' &\leq t \ \} \end{aligned} \tag{5}$$

With those two functions defined, we can now calculate the weight $w_u^{(i)}$ of a given member u during the recalculation iteration i. The weight is calculated as the percentage of members who all voted for the same action.

$$w_u^{(i+1)} = \alpha w_u^{(i)} + (1-\alpha) \frac{|votedFor(i \cdot d, latestVote(u))|}{\sum_{\theta \in \Theta} |votedFor(i \cdot d, \theta)|} \tag{6}$$

Where $w_u^{(0)} = 0.5$, an initial value chosen to give fair weighting when little is known about the member's reliability. Here, α is a historical decay rate to reduce the influence of previous agreement and ensure the weighting represents a recent agreement history. Then, whenever a vote is received, the action chosen by the agent is calculated using the following function:

$$f_{vote}(U, V, t) = \arg\max_{\theta \in \Theta} \sum_{u \in votedFor(t,\theta)} w_u^{(\lfloor t/d \rfloor)} \tag{7}$$

Where $\lfloor t/d \rfloor$ calculates the latest iteration index at current time t. For each action, the weights for the members that voted for it are summed and the action with the highest sum will be the action executed by the agent.

4.1.4 Real-Time Borda

The Borda count election method is often used in social choice theory [6]. Borda count requires voters to rank candidates in order of preference, giving 1 point to the least preferred candidate, 2 to the second least, and so on. The

candidate who receives the most points wins the election. It is considered a fairer method than majority voting because it can sometimes elect broadly acceptable candidates, instead of those preferred by the majority. Hence, in the context of CrowdDrone, the *Real-Time Borda* aggregator can be used to reduce the ability of malicious members, who may have the majority, to influence the election. Instead, because the most-voted for action is not always the winner, more broadly acceptable actions voted for by the well-meaning crowd may win the election.

Given that crowd robotics requires real-time input from crowd members, it is not feasible to elicit rankings of actions in order of the crowd member's preference. Instead, from a single vote we can assume a natural ranking. For example, if the member intends the agent to move forward ($\uparrow$), then we can assume that they rank forward ($\uparrow$) as their top choice, and since backward ($\downarrow$) is the opposite action, it would rank as their least desired action. However, little can be inferred, from just a single vote, about the actions in-between these, the actions of yawing left ($\leftarrow$), right ($\rightarrow$), or performing a new element of the action set, no action at all ($\emptyset$), and would therefore all be given the same ranking. We introduce $\emptyset$ to denote an instance of 'no action', so that if there is divided consensus amongst the crowd, the best action may be to remain stationary. This ranking is denoted as $\phi_u \in \Phi^{(t)}$, where $\Phi^{(t)}$ is the set of all members' votes in the past d seconds from the current time t converted into rankings.

Additionally, due to the nature of robotic control, we can refine this method further. As in Section 3, CrowdDrone uses a simplified set of actions for input, but this does not mean that the output set must also be reduced. The robotic agent can perform both linear and angular components in a single action at the same, such as moving forward while rotating left. With this in mind, the *Real-Time Borda* aggregator allows us to calculate a member's ranking for both the linear action set and the angular action set. We can then run two elections for both linear and angular motion and combine the two winning components into a single expanded action. The robotic agent will then perform a combined linear and angular action. Until now, the use of this expanded output set has been impractical for all other aggregation methods. The *Leader* aggregator can only forward the reduced inputs of the leader, and a *Real-Time Majority* vote on both linear and angular component actions would ensure that both elections always select an angular and a linear action combined. Whereas, *Borda* ranked voting enables us to sometimes choose inaction, $\emptyset$, for a single component (e.g., to apply only a forward linear action, but have no angular component).

To denote this increased set of output actions, we introduce a set $\vec{\Theta}$ of vectors comprising of a linear component and an angular component.

$$\vec{\Theta} = \{\uparrow, \emptyset, \downarrow\} \times \{\leftarrow, \emptyset, \rightarrow\}$$

Thus, we need to convert a crowd member's vote θ into two rankings, a linear and angular ranking. We represent this as a vector ϕ_u, which is comprised of a linear ranking and an angular ranking (see Equation 8). For example, if a crowd member votes $\uparrow$, the vote is translated to the two rankings $\big\langle\langle 3,2,1\rangle, \langle 1,2,1\rangle\big\rangle$. The first linear ranking shows the member prefers $\uparrow$ to $\downarrow$, and the second angular ranking shows preference for no angular action $\emptyset$, rather than turning.

$$\phi(\theta) = \begin{cases} \big\langle\langle 3,2,1\rangle, \langle 1,2,1\rangle\big\rangle & \text{if } \theta = \uparrow, \\ \big\langle\langle 1,2,3\rangle, \langle 1,2,1\rangle\big\rangle & \text{if } \theta = \downarrow, \\ \big\langle\langle 1,2,1\rangle, \langle 3,2,1\rangle\big\rangle & \text{if } \theta = \leftarrow, \\ \big\langle\langle 1,2,1\rangle, \langle 1,2,3\rangle\big\rangle & \text{if } \theta = \rightarrow \end{cases} \tag{8}$$

Real-Time Borda then calculates the sum of the rankings, $\bar{\phi}^{(t)}$, for each possible candidate vector $\vec{\theta} \in \vec{\Theta}$, from every ranked vote as follows:

$$\bar{\phi}^{(t)}_{\vec{\theta}} = \sum_{\phi_u \in \Phi^{(t)}} \phi_{u\vec{\theta}} \tag{9}$$

The chosen action performed by the robotic agent is then the candidate with the greatest sum of points, as computed by f_{borda} (see Equation 10).

$$f_{borda}(U, V, t) = \arg\max_{\vec{\theta} \in \vec{\Theta}} \bar{\phi}^{(t)}_{\vec{\theta}} \tag{10}$$

4.2 Comparison

Now that the four methods have been introduced, here we discuss the relative strengths and weaknesses of each aggregation method. In more detail, in our applications, operating in real time and responding to real-time changes is critical. The *Mob* aggregator is likely to remain the most responsive to real-time changes, because it instantly performs the latest vote. Then the *Real-Time Majority* and *Real-Time Borda* both recalculate the best action to perform after every new vote is received, ensuring that their output is always up to date with the crowd's opinion. Finally, the *Leader* aggregator aims to be responsive to real-time events because, with a single user in charge, there is no deliberation required. Provided the leader reacts quickly, so too will the agent. Furthermore in situations of uncertainty, for example a situation where two equally viable paths are presented, the *Mob*, *Real-Time Majority* and *Real-Time Borda* aggregators could struggle when the crowd's intentions are divided, but the *Leader* aggregator has only one person in control and thus no divided consensus.

These aggregators, however, come with a number of shortcomings. For example, the *Mob* aggregator does not consider the crowd members' reliability or whether the rest of the crowd agrees with the vote. Hence, some instantly applied votes may contradict the previous vote, resulting in 'thrashing', where the agent performs an action and shortly after, performs the opposite action. Furthermore, this method is susceptible to attacks where even a single malicious member can undermine the entire system [12]. Thus, while *Mob* may perform well in ideal conditions, in applications where the reliability of the crowd is not controlled, *Mob* is unlikely to perform well. Conversely, the *Leader* aggregator chooses only the most agreed-with crowd members, and thus is less susceptible to malicious crowd members. However, the aggregator can only ever select the most average crowd member for direct control. As a consequence, the control scheme can never achieve a greater performance than that of the average individual, who may be too slow to react in real time. On the other hand, the *Real-Time Majority* and *Real-Time Borda* are required to balance real-time reactions with reliability. Enough people must first vote for an action before

it happens, whereby there is a delay between the first vote for an action, and it eventually being acted upon.

Next, we describe the method and results of the empirical evaluation of the aggregation methods detailed in this section.

5. EMPIRICAL EVALUATION

The evaluation of the real-time consensus methods, described above, allows us for the first time to determine which methods of aggregation are suitable for crowd robotics in a real-world experiment. Furthermore we aim to determine if some methods are more suited to specific scenarios, such as a dynamically changing environments.

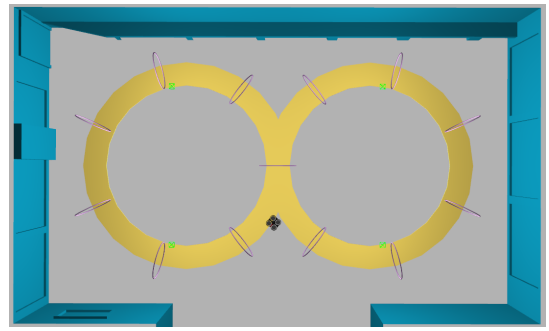
The following experiments uses a real-time simulation environment (Section 5.1) capable of realistically simulating the dynamics of the agent in a given scenario, and a simultaneous crowd of voters with an interface for shared control (bottom right of Figure 1), in which the crowd can observe the simulated agent and then vote on actions they wish the agent to take. It was deemed necessary for CrowdDrone to use a simulated environment for the practical reason that performing experiments with real-time crowds from AMT is time-consuming, and that it enables us to quickly create dynamic environments. To ensure fair benchmarking, we keep the user interface and the dynamics of the simulated agent the same, only varying the method of input aggregation and scenario.

When hiring the real-time crowd from AMT, the crowd was instructed in how to control the UAV with a pictorial diagram (seen in Figure 1). Short bullet points informed them that others would also be controlling the drone and that they should ensure the flight remains sensible. However, the crowd was not directly incentivised to perform the tasks correctly. Instead, they were informed only that they would receive payment provided that they vote, regardless of whether those votes were beneficial.

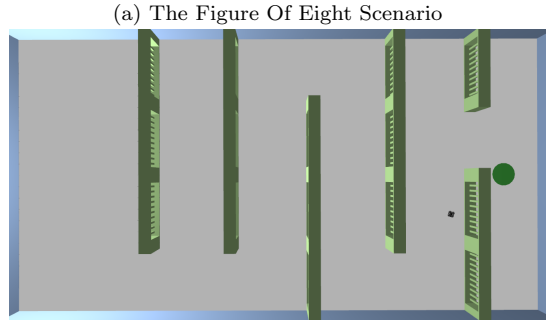
For these experiments, the constant parameter, d, used by the *Leader*, *Real-Time Majority*, and *Real-Time Borda* aggregators, denotes the election duration. This parameter determines how long the aggregators wait before considering members' votes no longer relevant. If the duration is too short, we would disregard many members' votes and then our calculation of the crowd's intentions would be biased towards the faster crowd members. However, if d is too long, then the aggregation algorithm may consider outdated votes. Outdated input may no longer be relevant if the robotic agent has since moved far from the initial location of the vote. We empirically chose the duration to be 2 seconds long, due to pilot experiments showing that the mean crowd input rate is 40 votes per minute. An election with a duration of 2 seconds ensures that we are capturing most of the members' latest votes when calculating a consensus.

5.1 Real-time Environment

As discussed in Section 3, CrowdDrone uses a simulated environment as it enables the fast development of dynamic scenarios. The environment used for the evaluation in this paper is bounded by visually distinctive walls that serve two purposes. Firstly, this prevents the drone from leaving the experimentation area. Secondly, it provides the users with an easy means to orientate themselves in the world. Inside this environment we can envision a number of scenarios that provide a challenge for calculating group consensus. For



(a) The Figure Of Eight Scenario



(b) The Dynamic Blocked Path Scenario

Figure 2: The CrowdDrone testbed environments

this study, we have considered two scenarios. The first is a simple scenario designed to evaluate the performance of the aggregation algorithms (Section 5.1.1). The second is a scenario in which the crowd must respond to changes in the environment (Section 5.1.2).

5.1.1 The Figure of Eight Scenario

The *Figure of Eight* (*FOE*) scenario has a number of equidistantly spaced rings arranged in a figure of eight pattern. Thirteen rings are placed in total, six around each circular section, and one shared ring in the middle (see Figure 2a). The crowd then attempts to fly the drone through the rings. This provides a simple environment in which to test and measure the basic functions of crowd robotics. Given the uniformity of the layout of the rings, the time taken to fly between each ring is a simple metric to compare the aggregator methods against. Additionally, given that in this scenario we know the optimal path beforehand, we can also calculate a positional error from the optimal path to measure how much the aggregators deviate from it.

5.1.2 The Dynamic Blocked Path Scenario

The *Dynamic Blocked Path* (*DBP*) scenario presents the crowd with the end goal far away down a long empty corridor. They must then navigate down this corridor. However, at five equally spaced points down the corridor, a wall will appear directly in the path of the drone. This impedes their progress and they must navigate around this new wall, reacting dynamically to the new environment, and attempt to reach the end goal (see Figure 2b). The wall consists of four randomly creatable sections. The two sections closest to the drone will always appear, ensuring the crowd must react, and one of the remaining two sections is created randomly, to ensure the crowd cannot learn where the open section may appear. This scenario is designed to test the crowd's ability to react to changes.

5.2 Evaluation Metrics

In order to compare the various aggregation algorithms, we define the following metrics relevant to the field of crowd robotics.

1. **Time between rings:** this is a general measure of how well the input aggregation algorithm performs. A longer time taken to travel a fixed distance can mean that there are more contradictory movements, more hesitation, or that a less than optimal path was taken.

2. **Positional Error:** the average distance from the optimal path can show how well the input aggregation method is filtering out noise, and responding to real-time updates.

3. **Stationary time:** the time spent stationary per minute is a measure of when the drone is either rotating or stopped (i.e., action $\emptyset$ is chosen) because of divided consensus as in the case of *Real-Time Borda*.

4. **Inaction time:** the time spent, per minute, not performing any action can be an indicator of low crowd participation.

5. **Input rate:** the rate of users' inputs per minute. Higher input rates correlate with greater crowd engagement, and can also be indicative of disagreement with the agent's actions.

6. **Reaction time:** the time taken between the wall appearing, in the *DBP* scenario, and the drone reacting to the new environment. The drone is considered to have reacted when its heading has changed by at least 20 degrees in an attempt to change course and navigate around the newly spawned obstacle.

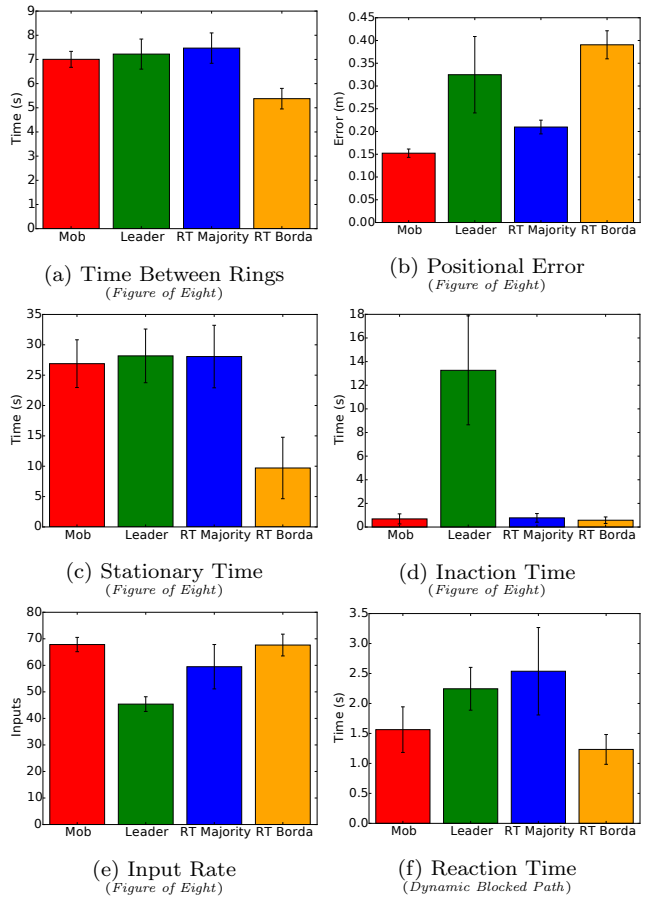
5.3 Results

In this section we evaluate the various aggregation methods (Section 4.1) as used in the two scenarios (Section 5.1). We ran 10 experiments for each aggregation method on each scenario. The real-time nature of these experiments is prohibitive to running large numbers of repeats. The total number of active crowd members (those who voted at least once), varied from 4 to 13 per experiment, and on average, 4.4 votes from unique crowd members were received every d seconds (i.e., 2 seconds). For the *FOE* scenario, we acquired on average 29 minutes of flight time, passing through an average of 254 rings, or over 1km of flight for each aggregation method.

5.3.1 Performance

The performance measured in the *FOE* scenario shows that the *Real-Time Borda* aggregator is significantly[8] faster at travelling between the rings, than other aggregators (Figure 3a), and 1.8 seconds faster than the *Leader* aggregator, the current state-of-the-art aggregation method for real-time control. However, despite this faster result, the *Real-Time Borda* aggregator shows a similar positional error (39cm) to that of the *Leader* method (Figure 3b). Instead, for positional accuracy, the *Real-Time Majority* aggregator is significantly closer to the optimal path (10 centimeters) than the

[8]All results reported as significant were confirmed with t-tests at a Bonferroni corrected significance level $p < \frac{0.05}{4}$ and all figures show 95% confidence intervals.



(a) Time Between Rings (*Figure of Eight*)
(b) Positional Error (*Figure of Eight*)
(c) Stationary Time (*Figure of Eight*)
(d) Inaction Time (*Figure of Eight*)
(e) Input Rate (*Figure of Eight*)
(f) Reaction Time (*Dynamic Blocked Path*)

Figure 3: Performance Graphs

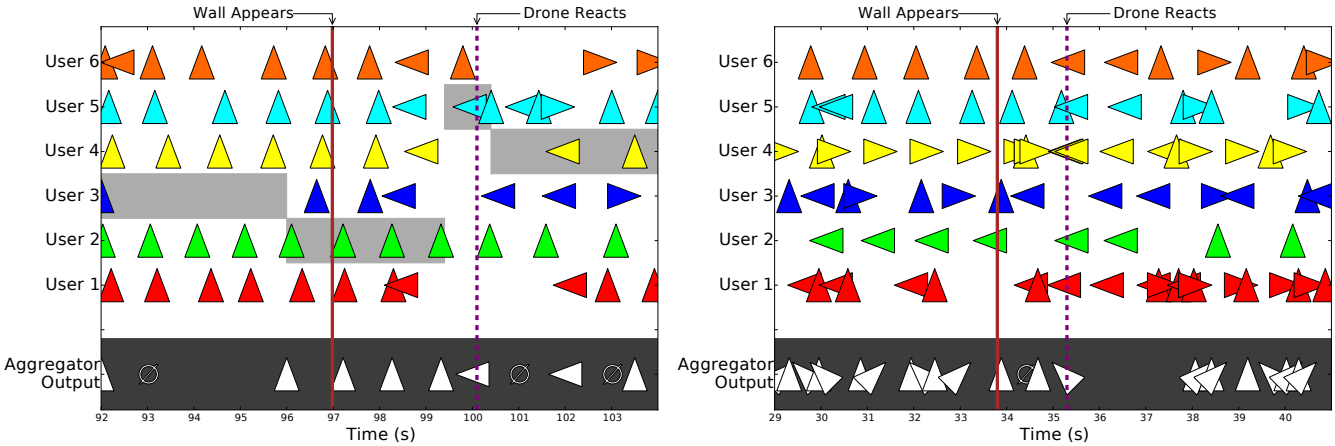
Leader aggregator. The *Mob* aggregator has even greater positional accuracy (15cm). This is because it instantly applies the crowd input, thus if it navigates off-course, a crowd member will quickly correct it. However, this is only true if there are no malicious crowd members. *Mob* has no method for reducing the influence of unreliable crowd members, and thus is susceptible to a single crowd member quickly and repeatedly voting for their own intentions, and thus navigating the drone off-course [12].

The *Real-Time Borda* aggregator also shows significantly less time spent stationary (9 seconds) than the others (Figure 3c), as the aggregator enables the drone to move and turn at the same time. Notably, the other aggregators spend nearly half the time rotating, which is expected on a circular path such as the Figure of Eight. The *Leader* aggregator remains inactive 12.5 seconds per minute, significantly longer than any other aggregator (Figure 3d), which is because control is given to a single operator, and the drone can only ever act as quickly as that operator gives commands.

These results show that if the application is likely to contain a flight path that requires a lot of turning (i.e., patrolling or surveying), then the *Real-Time Borda* saves time. Whereas if the task requires precision (i.e., package delivery), then the *Real-time Majority* aggregator would be a better choice.

5.3.2 User Input

The *Leader* aggregator has significantly fewer inputs per minute (Figure 3e), which is indicative of a lower crowd



(a) *Leader* Aggregator

(b) *Real-Time Borda* Aggregator

Figure 4: A short extract of real crowd input and the aggregated output.

engagement in these tasks. This effect may be explained by a lack of feedback. For many users their input 'appears' to have no effect, given that only a single user is controlling the robot, whereas, for other aggregation methods, the crowd's input has an appreciable effect on the drone's actions.

Figure 4 shows an example of typical crowd input for the *DBP* scenario centered around the moment a wall appears. We can see the clear differences in user input rate between the *Leader* aggregator and the *Real-Time Borda* aggregator.

5.3.3 Response to dynamic events

The reaction time is the time taken for the drone to react to an external change (see Figure 3f). The *Real-Time Borda* aggregator is significantly faster to react (1.2 seconds), the separated linear and angular preferential elections require fewer crowd members to vote for an action before the drone reacts. The *Mob* aggregator is also able to react quickly (1.6 seconds), as it instantly applies the input of the fastest crowd member, but slow crowd members can negatively affect its reaction time. Similarly, the *Real-Time Majority* can only react to the event when a majority of users have changed their vote, thus slow users can affect this aggregator too. The *Leader* aggregator, instead, only reacts as quickly as the crowd member currently in direct control of the UAV. If the leader reacts too slowly, the crowd begins to disagree with them, and another leader will be elected (see Figure 4a).

Figure 4 shows a comparison between the *Leader* and the *Real-Time Borda* aggregators, for the same *DBP* scenario with similar crowd sizes. At the far left of Figure 4a we can see the current leader highlighted with a grey background, User 3, but due to inaction control is then passed to User 2. When the wall appears, User 2 does not react, while the rest of the crowd shows disagreement with this, and thus control is again passed, now to User 5. It is the combination of low input rates and these slow to respond leaders, that make the *Leader* aggregator a poor choice for scenarios that require quick reaction times. In contrast, Figure 4b shows that the *Real-Time Borda* aggregator outputs many more actions per minute, often making many micro adjustments along the flight path. After the wall appears, we can see the drone begins turning even before the majority of users have reacted. This is due, in part, to User 1 and 3's fast reactions. Furthermore, we can see that User 2 has been consistently disagreeing with the crowd, therefore User 2's votes would have a lower weighting.

In general our results show that the *Leader* aggregator is often unsuitable for real-time control of a UAV. Instead, for applications that require speed, and quick reactions the *Real-Time Borda* aggregator performs best. On the other hand if the application requires accuracy, at the cost of speed, *Real-Time Majority* is a better choice.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced CrowdDrone as a testbed platform for evaluating various real-time aggregation methods of crowdsourced input. We then described two novel methods of real-time aggregation, *Real-Time Majority* and *Real-Time Borda*, and used CrowdDrone to evaluate their suitability for crowd robotic applications. Our empirical evaluation on a real-time crowd hired from AMT showed that the *Leader* aggregator, the state-of-the-art method, is not ideal for real-time control of a UAV. The input rate and reaction times of individual users are too slow for practical use. The *Mob* aggregator is also not usable for our applications, given that it does not filter unreliable users. Thus, malicious crowd members are capable of manipulating the UAV to their own ends. Instead, *Real-Time Majority* and *Real-Time Borda* provide faster or more accurate control while still reducing the impact of unreliable crowd members. The results show that *Real-Time Majority* is useful in applications where precision is required, and that *Real-Time Borda* is a faster alternative for patrolling and surveying applications.

There are a number of challenges that remain to be addressed. Given the costs (i.e., fuel, lost packages, damaged drones) associated with drone deployments, there is a need to measure the robustness of real-time aggregation methods to malicious crowd members. In future work, the CrowdDrone platform will be extended to measure just how reliable these aggregation methods are under noisy conditions, and provide performance guarantees for using the aggregators within a ratio of beneficial to malicious crowd members.

7. ACKNOWLEDGMENTS

This research was undertaken as part of the ORCHID project funded by the EPSRC (EP/I011587/1).

REFERENCES

[1] M. S. Bernstein, J. Brandt, R. C. Miller, and D. R. Karger. Crowds in Two Seconds: Enabling Realtime Crowd-powered Interfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 33–42, New York, NY, USA, 2011. ACM.

[2] M. L. Cummings, S. Bruni, and P. J. Mitchell. Human supervisory control challenges in network-centric operations. *Reviews of Human Factors and Ergonomics*, 6(1):34–78, 2010.

[3] R. D'Andrea. Can Drones Deliver? *Automation Science and Engineering, IEEE Transactions on*, 11(3):647–648, July 2014.

[4] W. S. Lasecki, K. I. Murray, S. White, R. C. Miller, and J. P. Bigham. Real-time Crowd Control of Existing Interfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 23–32, New York, NY, USA, 2011. ACM.

[5] E. Law and L. v. Ahn. Human computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(3):1–121, 2011.

[6] A. Mao, A. D. Procaccia, and Y. Chen. Social Choice for Human Computation. In *Proceedings of the Fourth Workshop on Human Computation (HCOMP-12)*, pages 136–142, 2012.

[7] S. Montambault, J. Beaudry, K. Toussaint, and N. Pouliot. On the application of vtol uavs to the inspection of power utility assets. In *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on*, pages 1–7, Oct 2010.

[8] D. W. Murphy and J. Cycon. Applications for mini VTOL UAV for law enforcement, 1999.

[9] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[10] D. Schulz, W. Burgard, D. Fox, S. Thrun, and A. B. Cremers. Web interfaces for mobile robots in public places. *Robotics Automation Magazine, IEEE*, 7(1):48–56, Mar. 2000.

[11] A. Sheshadri and M. Lease. SQUARE: A Benchmark for Research on Computing Crowd Consensus. In *HCOMP*, pages 156–164, 2013.

[12] N. Stefanovitch, A. Alshamsi, M. Cebrian, and I. Rahwan. Error and attack tolerance of collective problem solving: The DARPA Shredder Challenge. *EPJ Data Science*, 3(1):13, 2014.