Optimising Robotic Swarm Movement

Manish Kumar Bera
Mentor: Dr. Indranil Saha

July 7, 2017

1 Introduction

Swarm robotics is a field of multi-robotics in which large number of robots are coordinated in a centralized[1] or distributed[3] way. It is supposed that a desired collective behavior emerges from the interactions between the robots and interactions of robots with the environment. With recent technological developments, producing a large number of inexpensive robots that are equipped with sophisticated sensing, computation and communication tools has become a reality[5]. Swarm robots can be used to form various spatial arrangements for any required purpose. Robots need to move simultaneosly, to go from one position to another, to collectively form a new configuration. Describing complex spatial specifications for swarms is a non-trivial task.

2 Objective

Describing complex spatial specifications for swarms is a non-trivial task. Our goal is to optimise the cost of the collective motion of the robots from initial configuration to final configuration.

3 Methodology

We consider a grid A of size $n \times n$. We will refer the collection of all the robots in the grid as *swarm.* We will have R number of robots such that a robot will occupy a cell of the grid. No two robots can occupy the same cell of the grid, and a robot will occupy only a single cell of the grid. In a timestep, each robot can either move to an adjacent cell, or may remain in the same cell. We will refer to a *move* as the movement of a robot. A robot can move only to an adjacent cell upon executing a *move.* We define a *timestep* as the time period after which a robot can execute a *move.* A robot can execute a *move* on each *timestep.* Each move will be selected from a set of *Motion Primitives.* A robot may or may not execute a move on a *timestep.*

Let L be the total number of *timesteps* which the *swarm* takes to go from initial configuration to final configuration. Let x_t^i be a variable that describes the position of ith robot after t time steps. We define *swarm position at timestep* t, X_t as :

$$X_t = (x_t^1, x_t^2, x_t^3, ..., x_t^R)$$

i.e., X_t is an ordered collection of tuples, with the ith tuple describing the position of the ith robot after t *timesteps.* Thus, X_t describes the position of the *swarm* in the grid after t *timesteps.* So, we can say that $X_0 = (x_0^1, x_0^2, x_0^3, ..., x_0^R)$ represents the initial configuration of the *swarm*, and $X_L = (x_L^1, x_L^2, x_L^3, ..., x_L^R)$ represents the final configuration of the *swarm*.

The movement of the *swarm* will obey a set of rules. The progress from one *swarm position* to another will always follow certain conditions. Our aim is to develop a set of constraints, whose solution will give an optimal solution for the *swarm*. We will frame these constraints using a set of boolean variables, which will be encorporated into a formula ϕ. We define a *solution* S:

$$S = [X_0, X_1, X_2, ..., X_L]$$

where S is a sequence of *swarm positions* which occur in the chronological order of *timesteps*. The symbol S represents the sequence in which the *swarm* movement can occur from initial to final configuration. A *solution* S is said to be valid iff:

$$S \models \phi \tag{1}$$

$$\Leftrightarrow [X_0, X_1, X_2, ..., X_L] \models \phi \tag{2}$$

The problem will thus be reduced from an optimal path finding problem to a problem of finding satisfiability(SAT)[2]. We will use a SAT solver (like the MiniSAT[4]) to compute the solution to the above model.

4 Defining the constraints

The rules for the robots to move have to formulated into boolean expressions. For this we will have to define boolean variables.

4.1 Defining boolean variables:

I define $X(t, k, x, y)$ as a boolean variable s.t. $X(t, k, x, y)$ is *TRUE* iff at time step t, the kth robot is at position (x, y). the number of boolean variables will be $(L + 1) \times R \times n^2$

4.2 Constraints for initial and final states:

I define (x_0^k, y_0^k) to be the initial position of the kth robot. Similarly, (x_L^k, y_L^k) is the final position of the kth robot. The constraints for initial and final positions will be: $\forall t \in \{0, ..., L\}; \forall k \in \{0, ..., R-1\}$;

$$\begin{aligned} X(0, k, x, y) &= TRUE;\ if\ x = x_0^k\ and\ y = y_0^k \\ &= FALSE;\ otherwise \end{aligned}$$

$$\begin{aligned} X(L, k, x, y) &= TRUE;\ if\ x = x_L^k\ and\ y = y_L^k \\ &= FALSE;\ otherwise \end{aligned}$$

The time complexity for computing the constraints for the initial and final positions will be $O(R \times n^2)$.

4.3 Constraints for motion primitives:

I assume that the robots will have five simple motion primitives. each robot can:
1. Move up by one cell
2. Move down by one cell

3. Move right by one cell
4. Move left by one cell
5. Stay in that cell
Each robot has to perform one of the above mentioned motion primitives at each time step. Whether the robots are allowed to perform a particular primitive at a give timestep also depends on other constraints.

The constraints for motion primitives will be:
$\forall t \in \{0, ..., L\}; \forall k \in \{0, ..., R-1\};$

$$X(t+1, k, x, y) \Rightarrow X(t, k, x, y) \vee X(t, k, x-1, y) \vee X(t, k, x, y-1) \vee X(t, k, x+1, y) \vee X(t, k, x, y+1)$$

The time complexity for computing the constraints for the motion primitives will be $O(L \times R \times n^2)$.

4.4 Constraints for obstacle avoidance:

Let the number of obstacles be G. Let (x^g_{obs}, y^g_{obs}) be the position of the gth obstacle where $g \in \{0, 1, ..., G-1\}$. Each robot has to avoid each obstacle at every timestep. The constraints for obstacle avoidance will be:
$\forall t \in \{0, ..., L\}; \forall k \in \{0, ..., R-1\};$

$$X(t, k, x, y) = FALSE;\ if\ \exists g \in \{0, ..., G-1\}\ s.t.\ x = x^g_{obs}\ and\ y = y^g_{obs}$$

The time complexity for computing the constraints for obstacle avoidance will be $O(L \times R \times G)$.

4.5 Constraints for collision avoidance:

Each robot must avoid getting hit by another robot. There will be two types of collision.

4.5.1 Constraints for same space collision avoidance

This refers to the collision when two robots try to occupy the same cell of the grid. The constraints for this type of collision avoidance will be:
$\forall t \in \{0, 1, ..., L\};\ \forall k_1, k_2 \in \{0, 1, ..., R-1\}\ s.t.\ k_1 \neq k_2;\ \forall x, y \in \{0, 1, ..., n-1\};$

$$\neg(X(t, k_1, x, y) \wedge X(t, k_2, x, y))$$

The time complexity for computing the constraints for same space collision avoidance will be $O(L \times R^2 \times n^2)$.

4.5.2 Constraints for headon collision avoidance

This refers to the collision type where two robots move into each other. The constraints for this type of collision avoidance will be:
$\forall t \in \{0, 1, ..., L\};\ \forall k_1, k_2 \in \{0, 1, ..., R-1\}\ s.t.\ k_1 \neq k_2;\ \forall x, y \in \{0, 1, ..., n-1\};$

$$\neg((X(t, k_1, x, y) \wedge X(t+1, k_1, x+1, y)) \wedge (X(t, k_2, x+1, y) \wedge X(t+1, k_2, x, y))) \wedge$$

$$\neg((X(t, k_1, x, y) \wedge X(t+1, k_1, x, y+1)) \wedge (X(t, k_2, x, y+1) \wedge X(t+1, k_2, x, y)))$$

The time complexity for computing the constraints for headon collision avoidance will be $O(L \times R^2 \times n^2)$.

4.6 Soft constraints for cost optimization

For the optimisation of cost, we will need some additional constraints. Instead fomulating hard constraints, we formulate soft constraints, such that when the maximum number of those soft constraints are true, we will get the optimum cost. We will formulate each constraint such that it represents the negation of a particular motion. The soft constraints for cost optimization will be:

5 Implementation

For the implementation we use two SAT solver tools, namely, MiniSAT[4] and Open-WBO[6]. MiniSat is a minimalistic, open-source SAT solver. Open-WBO is an open source MaxSAT solver initially based on WBO. First we find the optimum *time length* for the problem using MiniSAT For solving the constraints, we transform the constraints into CNF form and feed it as input to the MiniSAT tool. For getting the correct value of L, we have to iterate over successive integers, starting from 1. For optimisation, I make exponential jumps, i.e., I iterate over 1, 2, 4, 8, ... After I get satisfiability, say at 2^a, I do a binary search between 2^{a-1} and $2^a$to get the least value of L for which the problem can be solved. I have also used other optimizaton techniques like reusing the constraints calculated for previous L values, instead of calculating them again.

6 Further Work

We wish to extend the developed implementation to include more complex motion primitives for smoother trajectories. We also wish to optimise the motion primitive cost using MAX-SAT. We further hope to enhance the compuation speed using techniques like multi-threading, and optimization methods such as pre-computation of constraints.

References

[1] Iman Haghighi, Sadra Sadraddini, and Calin Belta. *Robotic Swarm Control from Spatio-Temporal Specifications.* 2016 IEEE 55th Conference on Decision and Control (CDC), ARIA Resort & Casino, December 12-14, 2016, Las Vegas, USA. `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7799146`

[2] Wikipedia *Boolean satisfiability problem.* `https://en.wikipedia.org/wiki/Boolean_satisfiability_problem`

[3] Caroline Perry. *A self-organizing thousand-robot swarm.* `https://www.seas.harvard.edu/news/2014/08/self-organizing-thousand-robot-swarm`

[4] Niklas En, Niklas Srensson *The MiniSAT Page.* `http://minisat.se/Main.html`

[5] Self Organizing Systems Research Group. *The Kilobot Project.* `https://www.eecs.harvard.edu/ssr/projects/progSA/kilobot.html`

[6] Ruben Martins, Vasco Manquinho, Ins Lynce. *Open-WBO: An open source version of the MaxSAT solver WBO*
`http://sat.inesc-id.pt/open-wbo/index.html`