Hi my name is Karan Kamdar at MegaChess and today I am going to demonstrate to you an important project in the field of what we have defined as "Synergistic Social Robotics" that we believe has much to be thought about and some very important real world applications.

Outside on the Patio with large chess pieces:

Its been more than 2 really long years that I have been fascinated with a dream - a dream so large that I have been entirely absorbed in bringing it from the mind's conception to physical realization..... Do you see these large chess pieces? What if... these could move? Well, I am not talking about remote controlling them - far from it... What if these could move by themselves?.... Now that's just the start. What if... these could play the game of intelligence by communicating, sharing and arbitrating the best possible moves amongst themselves and then move autonomously? Wouldn't that be a dream worth spending 2 years for?....Well, what I shall demonstrate to you today is exactly this realization... I also come to a level of personal understanding where I believe that this project has applications to much more than what we had built it for. Now let's take you over inside to explain what this project is and how it works.

Inside:

What you see over here are 16 autonomous chess playing mobile robots with their own command and control ready too battle in a game of intelligence against 16 similar humans. Now your thought may be speeding away to .... "Well, where are the remote controls? Who's controlling them? Isn't there an external computer running the chess engine and sending commands to these bots?". Now that is what this is precisely not about. Instead what these robots are really architectured to do is to run their own independent chess engine stored in their tiny tiny microcontrollers, talk amongst themselves about what they are thinking, share best move vectors, aribtrate and finally make a choice of the best move that is communicated for visual purposes, through this (point at the computer) dumb, standalone, interface only computer judge to the humans. When I say this computer is dumb, then it literally is because there is no intelligence running on this machine, everything that is intelligent is inside these boys (well 15 boys and a girl:)). Now to walk you over through how these machines work, lets understand what's inside of them.

What's inside of them are two basic components - one is an entirely microcontroller embedded distributed chess engine and the other is an almost human-like communication framework that allows these robots that are P2P connected in a mesh network, which in other words means that each one of them can find the other... to communicate very useful and meaningful information. Now let's start with the chess engine. The chess engine uses the popular minimax search algorithm as its foundation. Now in chess engines that would use this algorithm, what would happen is that the processor would alternately play the role of a maximizer and a minimizer with the goal to minimize the maximum possible loss. To do this it would basically run through the search tree by generating a list of all possible moves. To generate a list of all possible moves ... it would basically have to loop through all the pieces, then all directions around each piece and finally all squares in those directions.

Now, in traditional single processor based chess engines this looping through of all possible moves would have to be done by that single processor. Instead what I have done over here is tapped into the power of multiple microcontroller processors to do two things - a. To speed up the process of move generation and best move evaluation and b. To avoid having to rely on a single machine to process the whole job. Now to give you an example, what would happen is that the king as the central aribtrator of selecting the final best move (which kind of makes sense because the game ends when the king is defeated) would initally send out a start signal to all the robots. Having received this start signal each of these robots would represent one of the 16 second level root nodes in the search algorithm and then each of these robots would compute moves that are pertaining to itself. Say for example the first pawn would generate moves and analyze them with itself as the center of the directional move genration process. Similarly for all the other robots. Finally each of these robots send their best evaluated moves back to the king who simply has to select the maximum value from the received best move vector. And with this... what happens is that the entire complexity of computation is equally distributed amongst all the robots at all times. Now that's the formulation of the core distributed chess engine which would probably be considered more like a star network between the king and other pieces had there not be the second component that I shall talk about.

And things really start getting exciting with this second component which is the human-like communication framework. This framwork operates at two basic levels. First at the level where each robot is doing the job of computing moves with itself as the center of the move generation process and... Secondly at the level where things like global map data sharing, or the knight's or the rook's move... where the pieces have to probe each other other and know who's where and adjust accordingly in order to make the desired move. Now in order to explain the first level at which this framework operates, lets for a little come back to the move generation process. Each robot actually goes a certain level of depth in the search tree in order to compute all possible moves. In order to initate robot communication at this level, what I have done is that made the depth level proportional to the battery charge that's remaining in the robot which in more human like form can be thought of how the tired the robot is or how actively interested it is in contributing to the swarm's decision.

So say the robot has a 90%-100% charge.. then it implements a full 4 levels of depth search. However if the robot is tired i.e. its battery charge is at say 60% then it would only go to 2 or 3 levels of depth. Now if the robot goes 4 full levels of depth then it sends out a "need_your_opinion" signal at an urgency value of 1 to all the other robots, which in other words means that "I don't necessarily need your opinion for this move but it would be nice to have it". In this case one or more of the other robots (depending on how much charge in remaining inside of them too) positions itself in the shoes of that calling robot and confirms if it has made the right calculations. In most cases, two robots implementing the same level of depth search should arrive at the same calculation.

However, let's take the situation where a more active, stronger robot with more battery charge and a greater level of depth search positions itself in the shoes of a tired robot at say 60 or 70% of battery charge which has sent out a "need_your_opinion" signal at an urgency value of 2. Now this is where true human-like P2P move aribitration and communication occurs. The stronger robot that goes a level 4 in the search tree would find itself at crossroads with the weaker robot and try to tell it that "Hey pawn, this move is better. What do you think?". In this case, where there is a potential of conflict between robots, then that weaker robot would call for the opinion of other robots which would ultimately help it to compute its desired value that it sends back to the king arbitrator.

Now at the second level where the communication framework operates is during certain situations in the game. Each robot maintains a record of its own position with respect to the overall game state and continuously updates a two way linked list data structure by P2P communicating with other robots that gives it knowledge of where the other pieces are with respect to itself. Take a situation where the first move in the game is a knight's move. In this case... this knight (pointing to the knight) is blocked by this guy P7 (pointing at P7) over here. In this case, the knight tells P7. Hey Mr. P7 you are blocking me.. P7 listens to the knight and adjusts accordingly relocating itself back to its original position once the knight's move is complete. Or take the more complicated castling move where more than three pieces have to adjust in order for the maneuver to be completed. In a castle move the king moves two squares to either of the rooks and the rook has to cross over to the king's skipped square. However the rook needs free space for this to happen. In many situations, the pawns ahead of the king and the rook have not moved and so the inter-robot communication goes from the king -> rook. If rook is blocked then it sends out an adjust signal to the blocking pieces which again relocate when the maneuver is complete. Lastly take the quit move, where the quitting piece calculates its position by getting position variables from all other surrounding pieces and then maneuvers itself out of the board. Each of these situations require a constant communication P2P between the robots.

This communication has to be however very reliable. In order to accomplish this, a packet confirmation and retry algorithm based on the popular TCP/IP protocal had to be implemented which makes sure that what the sender sent was indeed what the receiver received.

Contextualization

Now its important for me to contextualize this project within a field of research and this research field has been that of Group or Swarm Robotics. The idea of swarm robotics is not new. However, to quote Don Miner in his paper, "Swarm Robotics Algorithms: A Survey", when it comes to robotics applications, traditional Swarm Intelligence algorithms, such as Ant Colony Optimization (ACO) [1], do not transfer well to swarm robotics domains for several reasons. For example, ACO is hard to implement in a swarm robot because robots would have to drop pheromones and alter the environment, which is an unfavorable feature of robots systems and should be avoided. Meanwhile, dropping pheromones is not invasive at all in a computational world. This is a common theme when trying to transfer search algorithms and artificial intelligence techniques to robots. For example, breadth-first search is extremely inefficient when robots use it. A robot would have to backtrack and travel to that node instead of being able to move a pointer from node to node in a graph. For these reasons and others, devising algorithms for robot swarms has become a research field of its own".

Undoubtedly, Swarm robotics is a hot research area. However, Various algorithms for distributed problem solving are being developed, particularly in simulations. The focus is on complex, emergent behaviour arising from the local interactions of individuals following simple rules. There's nothing like reality to test out your robots, so research groups are beginning to invest in physical robot swarms. The cost of current robotics platforms prohibits experimentation with swarms numbering more than a few tens of units. As a result, the practicalities of software and hardware maintenance in large swarms are yet to be addressed

3. Current state of research -
   a. 100 iRobot Team
   b.

IRobot Swarm and SwarmBots Projects

Although this platform may be used to test similar Swarm robotics algorithms, the intention was to push the limits of emergent swarm intelligence to solve complex mathematical and real-world problems by using co-operative decision making. Whereas, a simple dispersion algorithm would involve minimal computation such as locating C closest neighbors and generating distance vectors to move away from each other, computing chess moves involves significant computational complexity such as looking ahead several levels of current game state, communicating and arbitrating move vectors with the swarm in order to select the best possible move. And a push towards such powerful processing capabilities is justified if have to see groups of truly intelligent and autonomous machines working with humans solving some of the toughest real world problems. For example, a robot that is deployed for real combat purposes will have to not only base its decision making on simple existing rules, but will need significant real-time decision making and communication capabilities to process and act on the combat information given its limited resources. And this has been the predominant reason behind building a hardware and software framework from the ground up, to address some of the issues that I feel need a proof of concept demonstration

Synergistic Robotics:
1.

User Experience

How the play would occur
Applications
Conclusion