SPATIO-TEMPORAL GESTURE RECOGNITION ON A DISTRIBUTED SENSING SYSTEM

by

AKSHAY RAGHUNANDAN MYSORE

B.Tech., National Institute of Technology, Durgapur, India, 2011

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirement for the degree of

Master of Science

Department of Electrical, Computer and Energy Engineering

2014

This thesis entitled:
Spatio-Temporal Gesture Recogniton on a Distributed Sensing system
written by Akshay Raghunandan Mysore
has been approved for the Department of Electrical, Computer and Energy engineering

_____
Dr. Nikolaus Correll (Committee chair)

_____
Dr. Richard Han (Committee member)

_____
Dr. Dirk Grunwald (Committee member)

Date_____

The final copy of this thesis has been examined by the signatories, and we
Find that both the content and the form meet acceptable presentation standards
Of scholarly work in the above mentioned discipline.

ABSTRACT

Mysore, Akshay Raghunandan (M.S., ECEE)

Spatio-temporal gesture recognition on a distributed sensing system

Thesis directed by Assistant Professor Nikolaus Correll

Gesture recognition using in-network processing rather than centralized processing is of importance in distributed sensor arrays embedded within systems such as tactile input devices, sensing skins for robotic applications, and smart walls. We have implemented a gesture recognition algorithm on a smart wall system with distributed sensing using capacitive touch sensors. Each node records a sense event and shares it with other nodes in the network to establish a sequence of sense events. Then each node builds up a chain vector array commonly used in gesture recognition algorithms and compares it with a reference gestures to find out the alphabet drawn on the wall. We studied the effect of gesture speed and number of vectors on gesture recognition and found that as the number of vectors in a gesture increases, the recognition becomes more reliable. A slower gesture speed allows for better sensing of touch and hence also leads to better gesture recognition.

We also implemented a Hardware abstraction Architecture based on the TinyOS operating system to achieve the goal of making the program platform independent.

ACKNOWLEDGEMENTS

Foremost, I would like to thank Prof. Nikolaus Correll for the continuous support he has provided me this past year. He helped me structure my project when I didn't know how and always pointed me in the right direction. His encouraging words and jovial attitude helped me in difficult times to keep working on this thesis.

I would like to thank Prof. Richard Han for initializing me into the research community at University of Colorado at Boulder by giving me the opportunity to work on numerous projects and finally on the Smart wall. I would have been lost without his mentorship.

I would also like to thank Prof. Dirk Grunwald for his encouragement and insightful comments as a part of my thesis committee. Due to our discussions on computer systems and coding, I was able to write good software and look at the system from a different perspective.

Last but not the least, I would like to thank Homa Hosseimardi for laying the groundwork for the gesture recognition algorithm and Nicholas Farrow for helping me get started on the hardware for the Smart wall.

CONTENTS

FIGURES

Figure

CHAPTER I

INTRODUCTION

Gesture-based control, ranging from simple directional swipes to input of complex characters has emerged as a standard method for human-computer interaction. Most systems however are spatially limited to sensor arrays, e.g., tactile displays, range-finding cameras such as the Xbox Kinect, and require information processing in a central unit. We wish to make gesture recognition available to a wider range of surfaces, which embed limited memory, computation and communication capabilities and can therefore function without a central processing unit. Examples of such surfaces range from a building wall made of smart bricks, robotic sensing skins, or smart paint [Butera et al].

In the short term, we are interested in smart building facades that allow users to control environmental conditions using gestural input without a central computer processing the data and independent of the arrangement of the smart bricks. An artist's impression of such a system as well as a distributed computing test-bed is shown in Fig 1 and Fig 2.

Fig. 1: Artist impression of an amorphous computational facade made of smart bricks and possible user interaction.

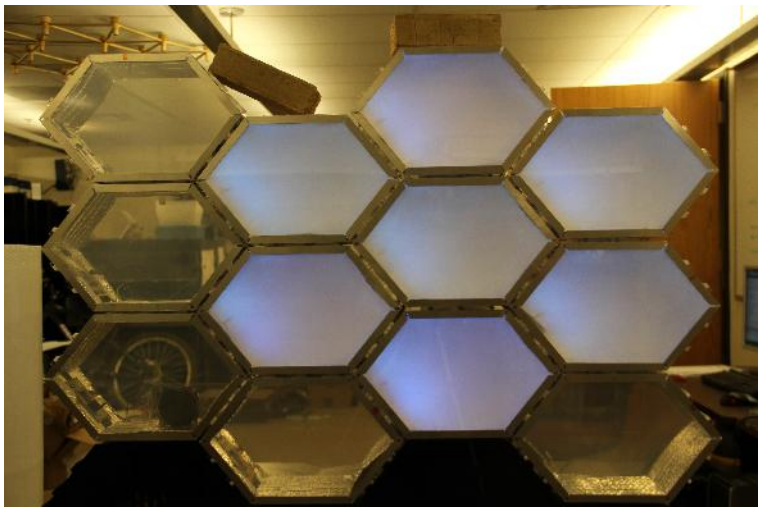

Fig. 2: Smart facade test-bed consisting of 12 cells and networked with their local neighbors. The color of cells change according to the input gesture.

Here, gestures would allow users to change the appearance of the wall, open a window or change the temperature. Similarly, when embedded in clothes [profita et al], gestures might allow communication with other devices such as music players or cell phones in an intuitive manner.

1.1 RELATED WORK

There exists a large body of work on gesture recognition and online handwriting recognition in touchpads, tablets, tablet computers [Wu and Huang 1999; Mitra and Acharya 2007] or other electronic devices [Davis and Lyall 1986]. We are not specifically interested in recognition of letters; stroke-based character recognition is a mature field and allows encoding and recognizing a large variety of gestures. These algorithms deal exclusively with a centralized representation of the gesture, that is gestures are recorded by a specific device such as a camera, range-finding device or inertial-measurement unit (IMU) [Schlömer et al. 2008], co-located with a computer and processed thereon. [Nagi et al. 2012] introduces a distributed decentralized gesture recognition algorithm to overcome the processing power limitation of swarm devices. In this work we aim to off-load both gesture recording

and processing to an input surface embedded with a sensor array without requiring a central computing device. This allows such a surface to be amorphous, scalable and potentially robust to failure of individual units.

A common approach to encode gestures/hand-written characters is to use a "chain code" [Ozer et al. 2001; Confer and Chapman 2004]. For this, consecutive points of a gesture are sampled at regular intervals, and a discrete number, e.g., 0-7 to encode angles in 45 degree intervals, will be assigned based on the angle of the two consecutive samples. The resulting number streams can then be classified, e.g., using Hidden Markov models [Ozer et al. 2001; Lee and Kim 1999;Wilson and Bobick 1999], support vector machines [Bahlmann et al. 2002] or neural networks [Murakami and Taguchi 1991].

In this work, we are building upon [Confer and Chapman 2004], which provides a simple algorithm for online alphabet recognition based on direction of writing. Unlike [Confer and Chapman 2004], which implements this algorithm on a touchpad device, we study extensions that allow us to divide the sensing load across a facade of distributed nodes.

CHAPTER II

MATERIALS AND METHODS

2.1 HARDWARE

A smart wall system was designed as a test bed for the distributed algorithms. It is composed of hexagonal blocks made of acrylic, 11.3 inches in width and 9.6 inches in Height. One Hexagonal face of the block is completely transparent whereas the other face is made of Polymer dispersed liquid crystal materials, also called smart glass. The PDLC material is generally opaque as the random alignment of the liquid crystal materials causes light to scatter, but when an electrical field is applied to them, they align in the same direction and allow light to pass through the material making it transparent.

Embedded in each of these blocks is a circuit board with an ATxmega128A3U processor which is a low power, high performance 8/16-bit AVR microcontroller featuring a 128KB self-programming flash program memory. This is the central computing unit of each block. The microcontroller is interfaced with many different peripherals such as a touch controller, accelerometer, RGB leds, LCDs and speakers. For the purpose of our experiment we use the Touch controller as the sensor, and the RGB leds along with the PDLC material as actuators. The touch controller is connected to 6 electrodes which are affixed on the six edges of the hexagonal block. These electrodes can be used as either touch sensors or proximity sensors with a sensing distance of around 6 inches.

Each block is connected to its neighbors via a UART port running at 115200 baud which acts as the communication channel for all the nodes in the system. Every block can communicate with every other block in the system.

2.2 SOFTWARE

2.2.1 Algorithm:

For our gesture recognition algorithm, we consider a distributed array of sensing nodes arranged as a lattice, with each node in the lattice having limited computational abilities and arbitrary, local connectivity to its immediate neighbors. We further assume that all nodes involved in a gestural event can communicate with each other by multi-hop multi-cast communication [Hosseinmardi et al. 2012; Ma et al. 2012]

As in centralized gesture recognition, pattern recognition will be based on comparison of an input stream with a dataset of sample vectors. A key challenge here is that gestures may have different sizes and therefore require scaling as a pre-processing step. In our scenario, gesture size is determined by the number of motion nodes as shown in Fig. 3. Therefore, data vectors inside motion node tables need to be interpolated (Fig. 4) or extrapolated to match the template length. Classification can then be performed by finding the template that minimizes a distance metric with the measurement vector.

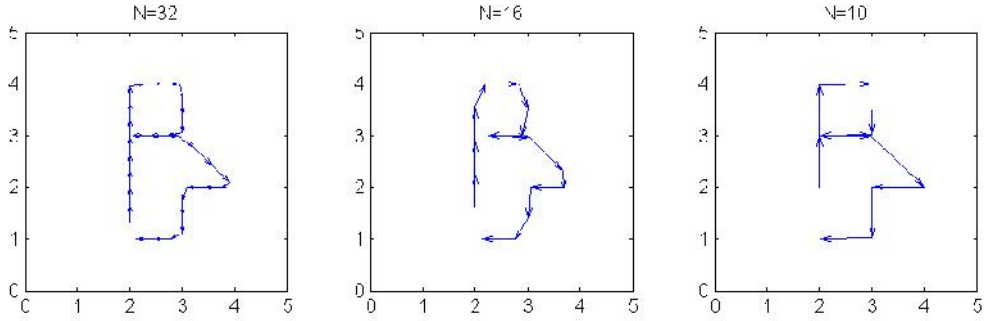

Fig. 3: chain vector for different sensor densities

In order to create a feature vector, a motion node emits a data packet upon a sensing event such as touch or light. Such an event packet conveys spatio-temporal information of a node that has detected an event, which can then be used to reconstruct a feature vector on each node receiving the packet.

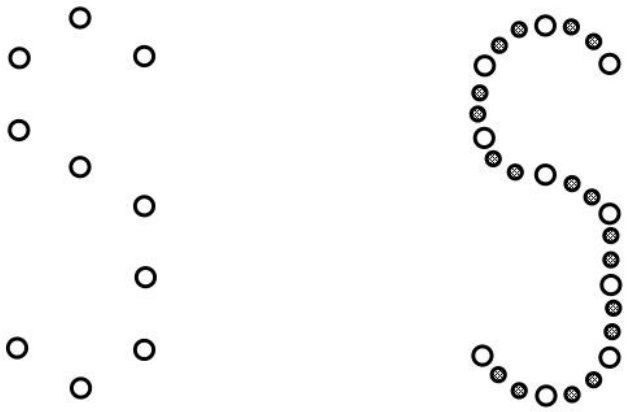

Fig. 4: Input curve of arbitrary size at a motion node (left). Interpolation to get fixed gesture length of N + 1 (right).

When someone draws a gesture, each motion node that senses the gesture also needs to learn about other nodes that sense the motion. After detecting an event, each motion node will share its spatial and order information with other motion nodes. Each node gathers the spatio-temporal information into a table that will be used later to extract feature vectors.

Because different directions of input gestures might have different meanings, all nodes need to know the order in which events were recorded. Nodes will sort member vectors and their spatial information based on their temporal order for the purpose of feature extraction. Each node 'm' will maintain two vectors on its table: (x-coordinate, y-coordinate).

We compute X_m and Y_m vectors, i.e the sequence of x-axis changes and y-axis displacements describing the gesture. These vectors will be compared in the next phase with the corresponding vectors in our dataset of possible gestures. Therefore they need a unique size, equal to the length of samples in the dataset. For this purpose each node m will interpolate X_m and Y_m vectors to achieve a fixed unique gesture length 'N+1'.

Each node m will retrieve a certain number of gesture points 'M' collecting shared packets by other nodes sensing the same gesture. Since some of the packets

might not be properly recorded or delivered, each node has a potentially different number of gesture points, and consequently a different view of the same gesture. We employ the chain technique to extract the feature in which each node will construct a fixed-length gesture consisting of 'N + 1' interpolated samples of the gesture points that it has collected. In this way, the fixed-sampling technique is fairly robust within limits both to how small a gesture is made, as well as to packet loss or sensor failure.

Considering that users may draw gestures with different sizes, or that different group members may have different numbers of nodes in their table (due to packet loss), input gestures are likely to have different vector lengths. For example, in Figure 3, any number of gestural sensing events may be collected at a particular node, but these are interpolated to produce an 'N+1' point gesture.

A gesture can now be represented by the 'N' vectors connecting the samples. We denote a chain of vectors recorded by node m as V^m.

Here, each element of V^m is calculated by the coordinates of the nodes involved.

$$v_x^i = (x^{i+1}-x^i) \text{ and } v_y^i = (y^{i+1}-y^i)$$

where i and i+1 are adjacent points in the sequence.

In order to recognize what gesture is made, we use a 1-nearest neighbor (1-NN) clustering algorithm to find the sample 's' that has the least distance from gesture 'm'.

Let $\Delta^{ms}=|V^m-V^s|$ be the distance between a measurement at node m and sample $s \in DS$, a dataset containing all samples. The function $f(V^m)$ will return the lowest distance of input gesture m from dataset samples s, such that

$$f(V^m)=\min|V^m-V^s|$$

Here, $|.|$ is the Euclidian distance between vectors.

The vector for the gesture of the sample dataset with the minimum Euclidean distance from the input is concluded as the gesture drawn on the wall.

2.2.2 Functional Requirements for algorithm:

In order to implement the gesture recognition algorithm we need to have the following mechanisms in place:

1. Co-ordinate system: To get the spatial location of the node
2. Synchronization between nodes: To determine the order in which events take place
3. Message passing: To share information with other nodes

Co-ordinate system: Since our system is reconfigurable, we had to make sure the coordinate system would work for any configuration of the system. When the nodes start up, they are devoid of coordinates. On waking up, each node queries neighboring nodes to find if they have been assigned coordinates, if they do, the querying node acquires the information from the queried node and informs its surrounding nodes of the change in its coordinates. Thus each node gets it's coordinates with reference to a node which already knows its coordinates. The origin can be hard coded as a node which will always be available in the system or can be set by a command from the user or can be set to be settled by the nodes themselves through arbitration for. In case of arbitration, the node with the smallest ID gets to be origin.

Synchronization: Synchronization is the means by which nodes determine the sequence in which the sensing events occurred. The nodes are connected to each other by means of UART running at 115200 baud. This speed is much faster than the speed at which a gesture can be drawn on the wall. Every time a touch electrode senses a touch, it sends out a message with the co-ordinate of the electrode and a sequence number. Whenever a node receives such a packet, it increments the

sequence no. and sends this number out as the sequence no. in case it senses a touch. Thus when the packets are arranged in the increasing order of their sequence number, they represent the gesture as a sequence of coordinate points.

Message Passing: Each node sends out a packet containing the sequence number and the coordinates of the electrode touched. These packets are transmitted using the UART to all the surrounding nodes, thus providing a multi hop path from every node to every other node in the system.

2.2.3 Hardware abstraction architecture:

In order to make our application platform independent we implemented a hardware abstraction layer. The hardware abstraction layer was based on the TinyOS model [Handziski et al. 2005] but simplified to increase ease of use and readability.

1. Hardware presentation layer: The lowest layer of the architecture which deals directly with the registers via get, set and clear functions. This was implemented as inline functions to reduce function calls and make the register access faster.
2. Hardware Adaptation layer: This layer provides functions which deal with a particular peripheral. It accesses peripherals with read_device, write_device functions which utilize the underlying layer as interface to implement communication protocols like I2C etc.
3. Hardware Interface Layer: This layer utilizes the interfaces provided by the Hardware Adaptation layer to provide generic interfaces to the hardware independent application code. For eg. Get_sensor_input, set_actuator, etc.

Using the interface provided by the Hardware interface layer, a user can write code which will work on any hardware, given that the underlying layers provide the right interfaces to the higher layers.

CHAPTER III

RESULTS

3.1 IMPLEMENTATION

The main body of the code is a loop in which the sensor is polled to check if it has sensed a touch. If there has been a touch, we get the coordinates of the sensor and increment the current sequence number. Then we combine the two into a packet and send it out to all the surrounding nodes (Appendix A). After sending out the packet, we start a counter to check if we receive a packet within 2.1 seconds. If no other sensor packet is received within 2.1 seconds, we send out a command packet to all the nodes involved in the gesture to start their computation. Since every node which is a part of the gesture will receive a packet from the next point in the gesture within 2.1 seconds, only the last node of the gesture will send out the command to start the computation.

For the sake of simplicity, as the packets are received, we store them in an array according to their sequence number. Once the start computation command is received, we calculate the sequence of vectors from the array of gesture points and store them in another array. Since in our system the adjacent sensors of adjacent blocks are encoded to the same coordinate, while calculating the vectors we make sure that the current point is different from the next point (Appendix B). We then call the interpolation function with the array of vectors as input. In the interpolation function (Appendix C), we find the magnitude of all vectors (square the x component and y component, add them, compute square root). Then we add the magnitudes together and divide it by the number of vectors required (number of vectors in gestures in the reference dataset). We divide the whole gesture into pieces of the resulting magnitude. To do that we divide each vector into smaller

vectors using the ratio of the original magnitude of the vector to the value we get by dividing the accumulated magnitude by number of vectors.

Once we have a gesture represented by the same number of vectors as the samples in the reference dataset, we compute the euclidean distance between corresponding vectors of the input data set and the reference sample (Appendix D). As we are computing the distance of the input from each sample, we keep a track of the lowest distance and the class that the corresponding sample lies in. Thus once we have calculated the distance with all the samples, we will have a decision regarding the gesture drawn on the wall. This decision is then sent out to all the other nodes in the system and the appropriate action associated with the particular gesture is taken, which in our case is lighting the appropriate led.

3.2 EXPERIMENTAL RESULTS

We implemented the gesture recognition algorithm on the Smart wall test bed to identify the alphabets 'C" and 'O' and a swiping gesture from right to left. Each alphabet has eight samples in the reference data sets against which every input gesture was compared. Each reference sample was composed of 10 vectors and the input gestures were interpolated to 10 vectors for comparison. Depending on the gesture identified by the system, we actuated the PDLC screen to open or close and the RGB leds to light different colors. We also retrieved the vector data of the input gesture to confirm our results.

After the gesture recognition algorithm was implemented on the Smart wall system, we tested it against different gesture speeds to test its reliability and different number of gesture points to simulate sensor failure and packet loss. Fig. 5 and Fig. 6 display the accuracy of the of the algorithm with respect to the time taken to draw the gestures.

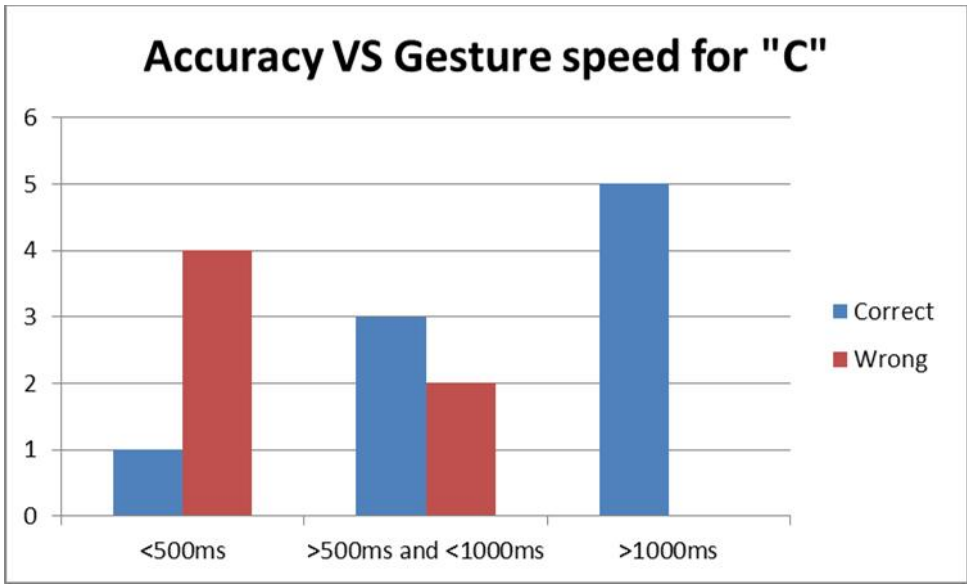

Fig. 5: Accuracy vs Gesture speed for 'C'

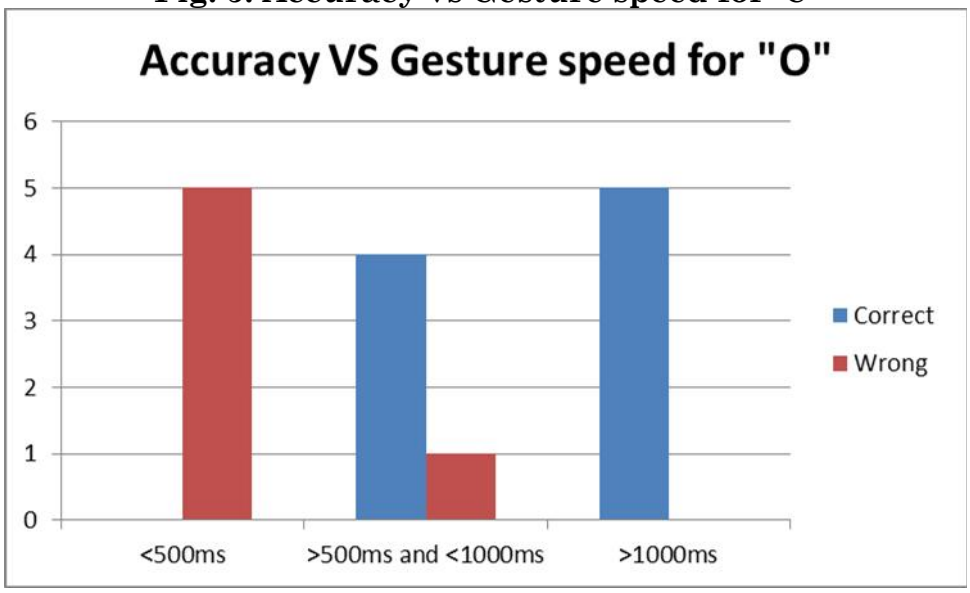

Fig. 6: Accuracy vs Gesture speed for 'C'

For determining the accuracy of the algorithm with respect to the gesture points, we selected the biggest vectors in the sample dataset for each alphabet and tested the output of the system for subsets of points in the gesture. The biggest vector for 'C' is composed of 5 points and the biggest vector for 'O' is composed of 6 points.

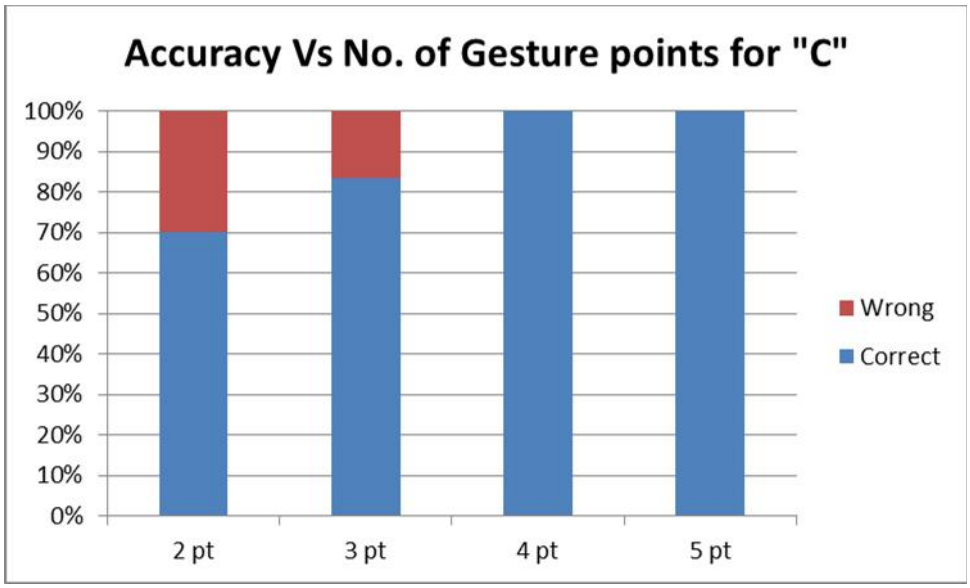

Fig. 7: Accuracy vs Number of Gesture points for 'C'

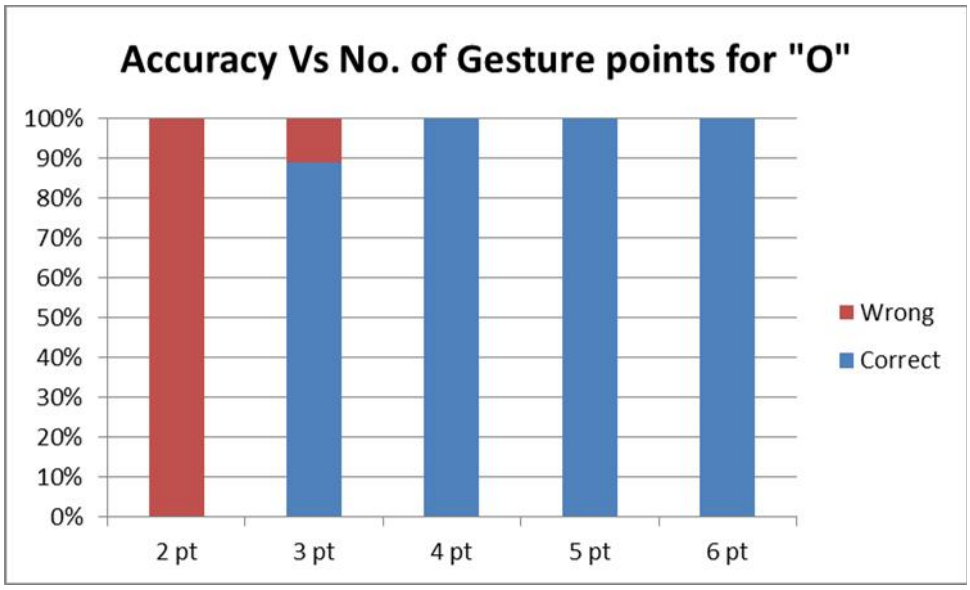

Fig. 8: Accuracy vs Number of Gesture points for 'O'

CHAPTER IV

DISCUSSION

In our experiments we found out that as the number of gesture points in an input gesture decreases, the gesture recognition reliability decreases. In our case, the alphabet C is mistaken for the sideswipe gesture. Since both the gestures start with the same direction, if the only points detected by the sensor are on a horizontal line, the algorithm classifies the input gesture into the sideswipe class. Thus the reliability of the algorithm increases when sufficiently different gestures are chosen. In the sample dataset, the gesture with the smallest length is the letter C. Thus most incomplete gestures are classified to be 'C' as it usually has the smallest Euclidean distance.

As for the speed with which a gesture is drawn, slower the gesture better is the recognition. In a slower motion, the sensors can more accurately detect if a touch has occurred. This leads to more number of gesture points in the input gesture and thus better recognition.

4.1 TIME COMPLEXITY

The algorithm can be divided into three main computation intensive stages 1) the sorting stage, 2) the interpolation stage and 3) the comparison stage. Of the sorting stage would have complexity O(nlogn), the interpolation stage would have complexity O(n) and the comparison stage would have complexity O(v*s), where n is the number of nodes in the gesture, v is the number of vectors in each sample gesture in the reference dataset and s is the number of samples in the reference dataset. Thus the time required for this algorithm is dependent on three factors, the number of nodes involved in the gesture, the number of vectors in each sample

gesture in the reference dataset and the number of samples in the reference dataset. We have observed in our experiments that more the number of nodes, better the recognition thus 'n' and 'v' will have the most effect for the computation time for a reliable distributed gesture sensing algorithm.

4.2 LIMITATIONS

1. Due to the large size of the blocks on the smart wall, smaller gestures are difficult to be identified due to lack of sufficient gesture points as shown in Fig. 9.

2. Multiple gestures cannot be drawn simultaneously, but this can be remedied by restricting the messages to only nodes part of the current gesture using Multicast groups or chain messaging.

3. The algorithm is not capable of detecting whether the input gesture is not a match to any gesture in the reference dataset. In our algorithm the gesture is always classified to its closest match. Therefore the alphabet 'G' will be classified to 'C' if the dataset contains only C, O and 4. For our current algorithm the reference dataset needs to contain samples of different sizes to reliably identify gestures of different size. This shortcoming can be eliminated by normalizing the size of all the gestures to the same gesture length.

4. As the node density increases, the chance that a number of nodes that may be excited at the same time increases. In such a case, our implementation will not be able to find the correct sequence of events as it depends on the gesture being slower than the communication between the nodes. In such a case, we will have to aggregate all the nodes excited at the same time to one particular point so that a proper gesture sequence can be derived from the gesture points.

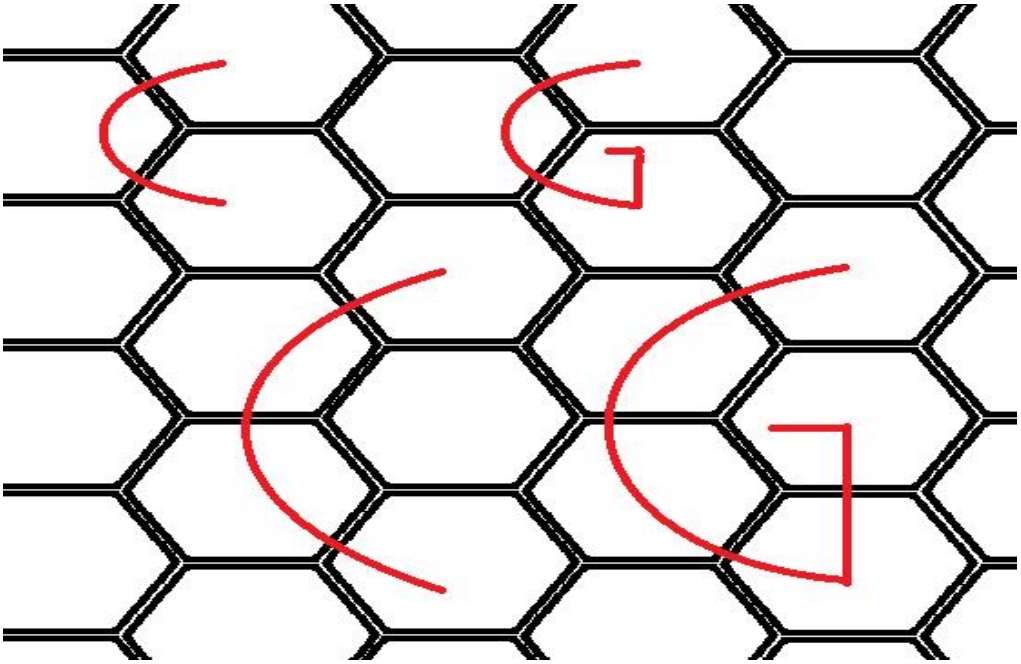

Fig. 9: Small gestures vs Large gestures

4.3 FUTURE WORK

We plan on implementing a bloom filter to help segregate multiple gestures drawn simultaneously. This will also help in faster identification of words as the user will not have to input one alphabet at a time.

The following algorithms still need to be tested on the smart wall.

1. Distributed Dataset: The reference dataset is divided such that each node has access to parts of the dataset through its neighbors. This reduces the memory requirement of the algorithm on each node.
2. Distributed Computation: Each node only operates on a part of the input vector and shares its results with the other nodes in the gesture to come to a decision. This reduces the computation required of each node.
3. Hybrid: Combination of Distributed dataset and Distributed computation

We are also planning on migrating from a touch based system to a touch less system so that a user does not have to be present next to a wall to interact with it.

CHAPTER V

CONCLUSION

In the course of this study we implemented a distributed gesture recognition algorithm on a real world distributed sensing system called the smart wall. The algorithm works by recording a sequence of sense events across the distributed system and then comparing it against a reference dataset to determine the gesture drawn on the system. Each node of the smart wall has six touch sensors, which on sensing a touch send out a packet with the sequence number and the location of the sensor. Then all the nodes involved in the gesture share these messages to get a table with all the gesture points. A vector table is then derived from these points and interpolated to match the number of vectors present in the dataset. The sample in the reference dataset with least Euclidean distance from the input gesture is determined to be the gesture drawn.

We then ran the algorithm with different gesture point sizes and at different motion speeds. We found that the reliability of the algorithm is directly proportional to input points and inversely proportional to gesture speed.

CHAPTER VI

REFERENCES

1. C. Bahlmann, B. Haasdonk, and H. Burkhardt. 2002. Online handwriting recognition with support vector machines-a kernel approach. In Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on. IEEE, 49–54.

2. W. Butera. 2002. Programing a paintable computer. Ph.D. Dissertation. Ph. D. dissertation, Program in Media Arts and Sciences, School of Architecture and Planning, MIT.

3. W.J. Confer and R.O. Chapman. 2004. System and method of handwritten character recognition. (April 13 2004). US Patent 6,721,452.

4. RH Davis and J. Lyall. 1986. Recognition of handwritten characters—a review. Image and Vision Computing 4, 4 (1986), 208–218.

5. H. Hosseinmardi, N. Correll, and R. Han. 2012. Bloom Filter-Based Ad Hoc Multicast Communication in Cyber-Physical Systems and Computational Materials. Wireless Algorithms, Systems, and Applications (2012), 595–606.

6. H.K. Lee and J.H. Kim. 1999. An HMM-based threshold model approach for gesture recognition. Pattern Analysis and Machine Intelligence, IEEE Transactions on 21, 10 (1999), 961–973.

7. S. Ma, H. Hosseinmardi, N. Farrow, R. Han, and N. Correll. 2012. Establishing Multi-Cast Groups in Computational Robotic Materials. In IEEE International Conference on Cyber, Physical and Social Computing (2012-11-20). Besancon, France.

8. S. Mitra and T. Acharya. 2007. Gesture recognition: A survey. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 37, 3 (2007), 311–324.

9. K. Murakami and H. Taguchi. 1991. Gesture recognition using recurrent neural networks. In Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology. ACM, 237–242.

10. Jawad Nagi, Hung Ngo, Alessandro Giusti, Luca Maria Gambardella, Jrgen Schmidhuber, and Gianni A. Di Caro. 2012. Incremental learning using partial feedback for gesture-based human-swarm interaction. In RO-MAN. IEEE, 898–905.

11. O.F. Ozer, O. Ozun, C.O. Tuzel, V. Atalay, and A.E. Cetin. 2001. Vision-based single-stroke character recognition for wearable computing. Intelligent Systems, IEEE 16, 3 (2001), 33–37.

12. H. Profita, N. Farrow, and N. Correll. 2012. In Adjunct Proceedings of the 16th International Symposium on Wearable Computers (ISWC) (2012-06-14). 44–46. http://correll.cs.colorado.edu/wp-content/uploads/ISWC2012 AdjunctProceedings.pdf

13. T. Schlömer, B. Poppinga, N. Henze, and S. Boll. 2008. Gesture recognition with a Wii controller. In Proceedings of the 2nd international conference on Tangible and embedded interaction. ACM, 11–14.

14. A.D. Wilson and A.F. Bobick. 1999. Parametric hidden markov models for gesture recognition. Pattern Analysis and Machine Intelligence, IEEE Transactions on 21, 9 (1999), 884–900.

15. Y. Wu and T. Huang. 1999. Vision-based gesture recognition: A review. Gesture-based communication in human-computer interaction (1999), 103–115.

16. V. Handziski, J.Polastre, J.H.Hauer, C.Sharp, A.Wolisz and D.Culler. 2005. Flexible Hardware Abstraction for Wireless Sensor Networks. In Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN 2005).

APPENDIX A

FORMATION OF GESTURE PACKET

```
gest_pkt pkt_g;
pkt_g.gest_coor = get_electrode_coordf(work_detail);
current_rank += 1;
my_rank = current_rank;
pkt_g.rank=current_rank;
Xgrid::Packet touch_pkt;
touch_pkt.type = MESSAGE_TYPE_GESTURE;
touch_pkt.flags = 0;
touch_pkt.radius = 6;
//generic_pkt.data = (uint8_t *)str;
touch_pkt.data = (uint8_t*)&pkt_g;   // 'data' field is of type uint8_t*
touch_pkt.data_len = sizeof(gest_pkt);
xgrid.send_packet(&touch_pkt);
points[my_rank-1]=pkt_g;
start_count=true;
gest_count=0;
```

APPENDIX B

CALCULATION OF VECTORS

```
for(iter=1,iter1=0;iter<current_rank;iter++){
        vec_in[iter1].vec_x = (double)(points[iter].gest_coor.x-points[iter-1].gest_coor.x);
        vec_in[iter1].vec_y = (double)(points[iter].gest_coor.y-points[iter-1].gest_coor.y);
        iter1++;
        if((vec_in[iter1-1].vec_y==0.0)&&(vec_in[iter1-1].vec_x==0.0)){
        iter1--;
        }
        printf_P(PSTR("%f\t%f\t%d\n\r"), vec_in[iter1-1].vec_x, vec_in[iter1-1].vec_y,iter1);
        _delay_ms(1);
        fflush(stdout);
}
```

APPENDIX C

INTERPOLATION

```
void interpolation (int num,vector* vec)
{
	int interp_num,interp_mod,i=0,j,k=0;
	double xx,yy,total=0,part,mag[SAMPLE_SIZE],rem;
	vector* vec_out_inter=vec_out;
	for(i=0;i<num;i++)
	{
		mag[i]=sqrt((vec[i].vec_x*vec[i].vec_x)+(vec[i].vec_y*vec[i].vec_y));
		total=total+mag[i];
	}

	part=total/(double)SAMPLE_SIZE;


	for(i=0;i<num;i++)
	{

		if(i!=0)
		{
		rem=part-rem;
		vec_out_inter->vec_x=(vec_out_inter->vec_x)+(vec[i].vec_x*rem/mag[i]);
		vec_out_inter->vec_y=(vec_out_inter->vec_y)+(vec[i].vec_y*rem/mag[i]);
		vec_out_inter++;
		mag[i]=mag[i]-rem;
		}
		rem=mag[i];
		for(j=0;rem>part;j++)
		{
			rem=rem-part;

		}
		interp_num=j;
		xx=vec[i].vec_x*part/mag[i];
		yy=vec[i].vec_y*part/mag[i];

		for(j=0;j<interp_num;j++)
		{
			vec_out_inter->vec_x=xx;
			vec_out_inter->vec_y=yy;
			vec_out_inter++;
```

```
                }
                vec_out_inter->vec_x=vec[i].vec_x*rem/mag[i];
                vec_out_inter->vec_y=vec[i].vec_y*rem/mag[i];

        }
        printf_P(PSTR("\n\r"));
        for(i=0;i<SAMPLE_SIZE;i++){
                printf_P(PSTR("%f,"),vec_out[i].vec_x);

                _delay_ms(1);
        }
        printf_P(PSTR("\n\r"));
        for(i=0;i<SAMPLE_SIZE;i++){
                printf_P(PSTR("%f,"),vec_out[i].vec_y);

                _delay_ms(1);
        }


}
```

APPENDIX D

EUCLIDEAN DISTANCE

```
if(euc_flag==1){
	result=0;
	for(i=0;i<14;i++)
	{

		if(i%2==0)
		{
			result_x=0;
			for(k=0;k<SAMPLE_SIZE;k++)
			result_x=result_x+((C[i][k])-vec_out[k].vec_x)*((C[i][k])-
vec_out[k].vec_x);
		}
		if(i%2==1)
		{
			result_y=0;
			for(k=0;k<SAMPLE_SIZE;k++)
			result_y=result_y+((C[i][k])-vec_out[k].vec_y)*((C[i][k])-
vec_out[k].vec_y);
			result=sqrt((result_x)+(result_y));
		if(i==1){
			least=result;
		}

		if(!(result>least)){
			count_vec=i/2;
			least=result;
			}

		}
	}
		for(i=0;i<16;i++)
		{
			if(i%2==0)
			{
				result_x=0;
				for(k=0;k<SAMPLE_SIZE;k++)
				result_x=result_x+((O[i][k])-vec_out[k].vec_x)*((O[i][k])-
vec_out[k].vec_x);
			}
			if(i%2==1){
				result_y=0;
```

```
                        for(k=0;k<SAMPLE_SIZE;k++)
                        result_y=result_y+((O[i][k])-vec_out[k].vec_y)*((O[i][k])-
vec_out[k].vec_y);
                        result=sqrt((result_x)+(result_y));
                        if(result<least){
                                count_vec=7+(i/2);
                                least=result;
                        }
                }
        }
        for(i=0;i<4;i++)
        {
                if(i%2==0)
                {
                        result_x=0;
                        for(k=0;k<SAMPLE_SIZE;k++)
                        result_x=result_x+((close[i][k])-vec_out[k].vec_x)*((close[i][k])-
vec_out[k].vec_x);
                }
                if(i%2==1)
                {
                        result_y=0;
                        for(k=0;k<SAMPLE_SIZE;k++)
                        result_y=result_y+((close[i][k])-vec_out[k].vec_y)*((close[i][k])-
vec_out[k].vec_y);
                        result=sqrt((result_x)+(result_y));
                        if(result<least){
                                count_vec=15+(i/2);
                                least=result;
                        }
                }
        }
        if(count_vec<7){
                set_rgb(0,255,0);
                printf_P(PSTR("\n\r\t\t\t\t\t<7\t\t\t\n\r"));
        }
        else if ((count_vec>7)&&(count_vec<15)){
                set_rgb(255,255,0);
                printf_P(PSTR("\n\r\t\t\t\t\t>7 <15\t\t\t\n\r"));
        }
        else{
                set_rgb(0,0,0);
                printf_P(PSTR("\n\r\t\t\t\t\t>15\t\t\t\n\r"));
        }
        euc_flag=0;
}
```