IntroductionToRobotics-Lecture09

Instructor (Oussama Khatib):Okay. Let's get started. So today it's really a great opportunity for all of us to have a guest lecturer. One of the leaders in robotics vision. A Gregory Hager from John Hopkins who will be giving this guest lecture. On Monday, I wanted to mention that on Wednesday we have the mid-term in class. Tonight and tomorrow we have the review sessions, so I think everyone has signed on for those sessions. And next Wednesday the lecture will be given by a former Ph.D. student from Stanford University, Krasimir Kolarov who will be giving the lecture on trajectories and inverse kinematics. So welcome back.

Guest Instructor (Gregory Hager): Thank you. So it is a pleasure to be here today and thank you, Oussama, for inviting me. So Oussama told me he'd like me to spend the lecture talking about vision and as you might guess that's a little bit of a challenge. At last count, there were a few over 1,000 papers in computer vision in peer reviewed conferences and journals last year. So summarizing all those in one lecture is a bit more than I can manage to do. But what I thought I would do is try to focus in specifically on an area that I've been interested in for really quite a long time. Namely, what is the perception in sensing you need to really build a system that has both manipulation and mobility capabilities? And so really this whole lecture has been designed to give you a taste of what I think the main components are and also to give you a sense of what the current state of the art is and, again, it's obviously with the number of papers produced every year declining the state of the art is difficult, but at least give you a sense of how to evaluate the work that's out there and how you might be able to use it in a robotic environment. And so really, I want to think of it as answering just a few questions or looking at how perception could answer a few questions. So the simplest question you might imagine trying to answer is where am I relative to the things around me? You turn a robot on, it has to figure out where it is and, in particular, be able to move without running into things, be able to perform potentially some useful tasks that involves mobility. The next step up once you've decided where things are is you'd actually like to be able to identify where you are and what the things are in the environment. Clearly, the first step toward being able to do something useful in the environment is understanding the things around you and what you might be able to do with them. The third question is, once I know what the things are how do I interact with them? So there's a big difference between being able to walk around and not bump into things and being able to actually safely reach out and touch something and be able to manipulate it in some interesting way. And then really the last question, which I'm not gonna talk about today, is how do I actually think about solving new problems that in some sense were unforeseen by the original designer of the system? So it's one thing to build a materials handling robot where you've programmed it to deal with the five objects that you can imagine coming down the conveyor line. It's another thing to put a robot down in the middle of the kitchen and say here's a table, clear the table, including china, dinnerware, glasses, boxes, things that potentially it's never seen before. It needs to be able to manipulate safely. That, I think, is a problem I won't touch on today, but I'll at least give some suggestions as to where the problems lie. So I should say, again, I'm gonna really breeze through a lot of material quickly, but at the same time this is a class, obviously, if you're interested in something, if you have a question, if I'm mumbling and you can't understand me just stop and we'll go back and talk in more depth about whatever I just covered. So with that, the topics I've chosen today are really in many ways from bottom, if you will, to top. From low-level capabilities to higher level. Our first computational stereo, a way of getting the geometry in the environment around you, feature detection and matching, a way of starting to identify objects and identify where you are, and motion tracking and visual feedback, how do you actually use the information from vision to manipulate the world. And, again, I think the applications of those particular modules are fairly obvious in robotic mobility and manipulation. So, again, let me just dive right in. Actually, how many people here have taken or are taking the computer vision course that is being taught? Okay. So a few people. For you this will be a review. You can, I guess, hopefully bone up for the mid-term whenever that's gonna happen or something. And hopefully I don't say anything that disagrees with anything that Yanna has taught you. But so what is computational stereo? Well, computational stereo quite simply is a phenomenon that you're all very familiar with. It's the fact that if you have two light-sensing devices, eyes, cameras, and you view the same physical point in space and there's a physical separation between those viewpoints you can now solve a triangulation problem. I can determine how far something is from the viewing sensors by finding this point in both images and then simply solving a geometric triangulation problem. Simple. In fact, there's a lot of stereo going on in this room. Pretty much everybody has stereo, although oddly about 10 percent of the population is stereo blind for one reason or another. So it turns out that in this room there are probably three or four of you who actually don't do stereo, but you compensate in other ways. Even so, having stereo particularly in a robotics system would be a huge step forward. Sorry for the colors there. I didn't realize it had transposed them. So when you're solving a stereo problem in computer vision there are really three core problems. The first problem is one of calibration. In order to solve the triangulation problem I need to know where the sensors are in space relative to each other. A matching problem. So remember in stereo what I'm presented with is a pair of images. Now, those images can vary in many different ways. Hopefully they contain common content. What I need to do is to find the common content, even though there is variation between the images, and then finally reconstruction. Once I've actually performed the matching, now I can reconstruct the three-dimensional space that I'm surrounded by. So I'll talk just briefly about all three of those. So, first, calibration. So, again, why do we calibrate? Well, we calibrate for actually any number of reasons. Most important is that we have to characterize the sensing devices. So, in particular, if you think about an image you're getting information in pixels. So if you say that there's some point in any image, it's got a pixel location. It's just a set of numbers at a particular location. What you're interested in ultimately is computing distance to something in the world. Well, pixels are one unit. Distances are a different unit. Clearly, you need to be able to convert between those two. So typically in a camera there are four important numbers that we use to convert between them to scale factors that convert from pixels to millimeters and two point two numbers that characterize the center projection in the image. So the good news for you is that there are a number of good toolkits out there that let you do this calibration. They let you characterize what's called the intrinsic or internal parameters of the camera. In addition, we need to know the relationship of the two cameras to each other. That's often called the extrinsic or external calibration. That's also something that you can get very good toolkits to solve. There's a toolkit for MATLAB, in fact, that solves it quite well. So calibration really is just getting the geometry of the system. So we're set up and we're ready to go. Now, for the current purposes let's assume that we happen to have a very special geometry. So our special geometry is going to be a pair of cameras that are parallel to each other and the image claims are co plainer and, in fact, the scan lines are perfectly aligned with each other. So if I look at a point in the left image and if I wanted to find the same corresponding point in the right image at some physical point in the world it's gonna be on the same row and, in fact, that's gonna be true for all the rows of the camera. So it's a really convenient way to think about cameras for the moment. So for a camera system like that, solving the stereo problem really is from a geometric standpoint simple. So what did I say? I've got a point on one camera line; I've got a point in another camera, the other camera the same line. What I can do is effectively solve triangulation by computing the difference in the coordinates between those two points. Again, not to go into great detail, but I can write down the equations of perspective projection, which I've done here, for two cameras for what I call now the X coordinate. So, in fact, on this last slide I forgot to point it out, but I'm gonna use a coordinate system, in fact, through this talk, which is X going to the right, Y going down in the image, and, I guess, most of you should be able to figure out which direction Z goes once I've told you those two things, right? Where does Z go? Out the camera. So Z is heading straight out of the camera lens. So that's the coordinate system we're going to be dealing with. So the things that we can find fairly easily in some sense are the X and Y's of the point. The unknown is the Z. So whenever I say depth you can think of I'm trying to compute the Z's. So I can write down perspective projection for a left camera and a right camera, which I've done here. They are offset by some base line, which I've called B. I've also got the Y projection, but it turns out to be the same for both cameras because I've scan line aligned them. So I've got three numbers, XL, XR, and Y. I've got three unknowns X, Y, and Z. A little algebra allows us to solve for the depth Z as a function of disparity, which, again, is the difference between the two coordinates. The base line of the camera and this internal scaling factor, which allows us to go from pixels to millimeters. Just a couple things to notice about this. Depth is inversely proportional to disparity. So the larger the disparity the smaller the depth. Makes sense. I get closer like my eyes have to keep going like this more and more and more to be able to see something. It's proportional to baseline. If I could pull my eyes out of my head and spread them apart, I could get better accuracy and it's also proportional to the resolution of the imaging system. So if you put all that together you can start to actually think about designing stereo systems from very small to very large that operate at different distances and with different accuracies. The other thing to point out here is that depth being inversely proportional to disparity means that close to the camera we actually get very good depth resolution. As we get further and further away our ability to resolve depth by disparity goes down drastically. You see out here at a distance of ten meters one disparity level is all ready tens maybe even hundreds of centimeters of distance. So stereo is actually good in here. It's very hard out there. And, in fact, anybody happen to know the human stereo system what its optimal operating point is? Anybody ahead of class talk about that? It's right about here. It's about 18 inches from your nose. Right in this point, you have great stereo acuity. You get in here and your eyes start to hurt. You get out here past about an arm length and it just turns off. You actually don’t use stereo at long distances. You’re really just using it in this workspace and, of course, it makes sense. We’re trying to manipulate, right? Well, I made a strong assumption about the cameras. I said that they had this very special geometry. And back in the good old days when I was your age – those are the good days by the way just so you’re all aware of that fact. You’re in the good days right now. Enjoy them. Back in the good old days, we actually used to try to build camera systems that had this geometry. Because if you start to change that geometry you no longer get this nice scan line property. So, in fact, if I rotate the cameras inward, if I start to look at the relationship between corresponding points I start to get these rays coming up. So I pick a point here the line is now some slanted line. Well, it turns out luckily one of the things that really have been nailed down in the last decade or two is that that doesn’t matter. We can always take a stereo pair that looks like this and we can resample the images so it’s a stereo pair that looks like that and, in fact, by doing this calibration process we get it. So the good news is I can almost always think about the cameras being these very special scan line aligned cameras. And so everything I’m gonna say from now on is going to pretty much rely on the fact that I’ve done this so-called rectification process. So, again, a very nice abstraction. I’ll just point out, again I’m not gonna talk a lot about it, but the relationship that I just described. So how do I – if I have a point in one camera how do I know the line to look for it on the second camera? Well, it’s a function of the relationship, the location, and the translation between the two cameras. And it turns out, this is, again, something that’s really been nailed down in the last couple of decades; I can estimate this from images. So, in fact, I could literally take a pair of cameras, put them in this room, do some work, and I could figure out their geometric relationship without having any special apparatus whatsoever. What it also means is instead of doing stereo I could literally just take a video camera and walk like this, process the video images and effectively do stereo from a single camera using the video. And that’s because I can estimate this matrix E in that relationship up there, which if we worked out what E was it turns out to contain the rotation and translation between the two camera systems. And once I know that I can do my rectification and I can do stereo. So actually stereo is a special case in some sense of taking a video and processing it to get motion and structure at the same time. So, again, that’s a whole lecture we won’t go in there, but suffice to say that from a geometric point-of-view, we can actually deal with cameras now in a very general way with relatively little operator assumptions about how they start. Okay. So geometry calibration is done. I now want to reconstruct. But to do reconstruction I need to do matching. I need to look at a pair of images and say, hey, there’s a point here and that same point is over there and I want to solve the triangulation problem. So there are two major approaches, feature based and region based matching. So feature based, obviously, depends on features. I could run an edge detector in this room. I could find some nice edges, the chairs, the edge of people, the floor, and then I could try to match these features between images and now for every feature I’m gonna get depth. So if you do that you end up with these kind of little stick figure type cartoons here. So here it’s a set of bookshelves and this is a result of running a feature based stereo algorithm on that. So you can see on the one hand it’s actually giving you kind of the right representation, right? The major structures here are the shells and it’s finding the shells. But you notice that you don’t get any other structure. You’re just getting those features that you happen to pull out of the image. So the other approach is to say forget about features. Let me try to find a match for every pixel in the image. The so-called dense depth map. Every pixel's got to have some matching pixel or at least up to some occlusion relationships. So let's just look for them and try to find them. And so this is the so-called region matching method. I'm gonna actually pick a pixel plus a support region and try to find the matching region in another image. This has been a cottage industry for a very long time and so people pick their favorite ways of matching their favorite algorithms to apply to those matches and so on and so forth. So, again, a huge literature in doing that. There are a few match matrixes, which have come to be used fairly widely. Probably the most common one is something called the sum of absolute differences. It's right up there. Or more generally I've got something called zero mean SAD or sum of absolute differences. That's probably the most widely used algorithm. Not the least of which because it's very easy to implement in hardware and it's very fast to operate. So you take a region and you take the difference. Take a region, take a region, take their difference, take their absolute value, sum those and assume if they match that that difference is gonna be small. If they don't match it's gonna be big. All the other matrix I have up here are really different variations on that theme. If you take this region, take that region, compare them, and try to minimize or maximize some matrix. So if I have a match measured now I can look at correspondence. Again, I get to use the fact that things are scan lined aligned. So when I pick a point in the left image or a pixel in the left image I know I'm just gonna look along a single line in the right image, not the whole image. So simple algorithm for every row, for every column, for every disparity. So for every distance that I'm gonna consider essentially. I'm now gonna compute my match metric in some window. I'll record it. If it happens to be better than the best match I've found for this pixel – I'm sorry. If it's better than the best match I've found for this pixel I'll record it. If not I just go around and try the next disparity.

So you work this out what's – how much computing are you doing? Well, it's every pixel, rows, and columns, every disparity. So, for example, for my eyes, if I'm trying to compute on kind of canonical images over a good working range, you know, three feet to a foot, maybe 100 disparities, 120 disparities. So rows, columns, 120 disparities, size of the window, which might be, let's say, 11 by 11 pixels is a pretty common number. So there's a lot of computing in this algorithm. A lot, a lot of computing. And, in fact, up until maybe ten years ago even just running a stereo algorithm was a feat. Just point out that it turns out that the way I just described that algorithm, although intuitive, is actually not the way that most people would implement it.

There's a slightly better way to do it. This is literally MATLAB code, so for anybody in computer vision this is the MATLAB to compute stereo on an image. You can see the main thing is I'm actually looping first over the disparities and then I'm looping over effectively rows and columns, doing them all at once. The reason for doing this without going into details is you actually save a huge number of operations in the image. So if you ever do implement stereo, and I know there are some people in this room who are interested in it, don't ever do this. It's the wrong thing to do. Do it this way. It's the right way to do it. If you do this you can actually get pretty good performance out of the system.

Just one last twist to this. So one of the things that's gonna happen if you ever do this is it's gonna – you're gonna be happy because it's gonna work reasonable well, probably pretty quickly. And then you're gonna be unhappy because you're gonna see it's gonna work well in a few places and it's not gonna work in other places. Someone in computer vision can give me an example of a place where this algorithm is not gonna work or is unlikely to work? A situation? A very simple situation.

Student:[Inaudible]

Guest Instructor (Gregory Hager): Well, if the – but I'm assuming rectification.

Student:[Inaudible]

Guest Instructor (Gregory Hager): Yeah. The boundaries are gonna be a problem, right? Because I'm gonna get an occlusion relationships will change the pixels so that good. Occlusions are bad. How about this? I look at a white sheet of paper. What am I going to be able to match between the two images? Almost nothing, right? So you put a stereo algorithm in this room it's gonna do great on the chairs, except at the boundaries. But there's this nice white wall back there and there's nothing to match. So it's not gonna work everywhere. It's only going to work in a few places. How can you tell when it's working? It's one thing to have an algorithm that does something reasonable. It's another to know when it's actually working.

The answer turns out to be a simple check is I can match from left to right and I can match from right to left. Now, if the system's working right they should both give the same answer, right? I can match from here to here or here to here. It shouldn't make any difference. Well, when the right – when you have good structure to the image that will be the case, but if you don't have good structure to the image it turns out you usually are just getting random answers and so I said something like 100 disparities, right? So the odds that you picked the same number twice out of 100 is really pretty small. So it turns out that almost always the disparities differ and you can detect that fact and it's called the left-right check.

On a multicore processor it's great because you have one core doing left to right and one core doing right to left. At the end they just meet up and you check their answers. Here's some disparity match. These are actually taken from an article by Cork and Banks some years ago just comparing different metrics. Here I just happen to have two. One is so-called SSD, semi-squared differences. The other is zero mean normalized cross-correlation. A couple of just quick things to point out. So, like, on the top images you can see that the two metrics that they chose did about the same. You can also see that you're only getting about maybe 50 percent data density. So, again, they've done the left-right check. Half the data is bad.

So kind of another thing to keep in mind. Fifty percent good data. Not 100 percent, not 20 percent either. Fifty percent good data maybe. Second row, why is there such a difference here? Well, there's a brightness difference. I don't know if you can see, but the right image is darker than the left. Well, if you're just matching pixels that brightness difference shows up as just a difference it's trying to account for. In the right column they've taken so-called zero meaning. They've subtracted the mean of the neighborhood of the pixels. Gets rid of brightness differences, pulls them into better alignment in the brightness range, and so they get much better density. This is just an artificial scene. Pick the most interesting thing there is it's an artificial scene. You've got perfect photometric data and it still doesn't do perfectly because of these big areas where there's no texture. It doesn't know what to do. So it's producing random answers. The left-right check throws it out.

Student: So left-right check does not have any one guarantee?

Guest Instructor (Gregory Hager): There are no guarantees. That's correct.

Student:[Inaudible]

Guest Instructor (Gregory Hager): It could make a mistake, but the odds that it makes a mistake it turns out are quite small. It actually is – I should say it's a very reliable measure because usually it's gonna pick the wrong pixels. Occasionally, by chance, it'll say yes, but those will be isolated pixels typically. Usually if you're making a mistake you don't make a mistake on a pixel. You make a mistake on an area and what you're trying to do is really kind of throw that area away. But a good point. It will – it's not perfect. But actually in the world of vision it's one of the nicer things that you get for almost free. I just say that these days a lot of what you see out there is real time stereo. So starting really about ten years ago people realized if they're smart about how they implement the algorithms they can start to get close to real time.

Now you can buy systems that run pretty much in software and produce 30 frames a second stereo data. Now, it's kind of just a mention that – I said that the data is not perfect. Well, the data's not perfect and as a result if you could imagine what we'd like to do is take stereo and build a geometric model. All right. I'd like to do manipulation. I want to take this pan and I want to have a 3-D model and then I want to generate or manipulate 3-D. Well, you saw those images. Stereotypically doesn't produce data that's good enough to give you a high precision completely dense 3-D models.

It's an interesting research problem and, in fact, people are working on it, but it's a great modality if you just want to kind of rough 3-D description of the world. And you can run it in real time and you're now getting kind of real time coarse 3-D. And so I think that's why right now real time stereo is really getting a lot of interest because it gives you this coarse wide field, 3-D data, 30 frames a second. It lets you just do interesting things. I just thought I would throw in one example of what you can do. So this is actually something we did about ten years ago, I guess. You're a mobile robot and you want to detect obstacles and you're running on a floor. So what's an obstacle? Well, it's positive obstacle is something that I'm gonna run into or it could be a negative obstacle. I'm coming to the edge of the stairs and I don't want to fall down the stairs. So I'm gonna use stereo to do this and so the main observation is that since I assumed I'm more or less operating on a floor I've got a ground plane. And it turns out plainer structures in the world are effectively plainer structures in disparity space. So it's very easy to detect this big plane and to remove it and once you've removed that big plane anything that's left has gotta be an obstacle, positive or negative. It doesn't matter what it is. So you run stereo, you remove the ground plane, if there's something left that's something you want to worry about avoiding.

So here's a little video showing this. So there's a real time stereo system. So, first, we put down something that's an obstacle, it shows up, here's something which if you were just doing something like background subtraction you might say that that newspaper is an obstacle. It's different than the floor, right? So you would drive around it because it could be something you don't want to run into. But here since we've got this ground plane removal going, we can see that this is very clearly an obstacle. That is very clearly something that's just attached to the floor and disappears and you can go by it. And this is something real cheap, easy, simple to implement.

I think that really is the great value of stereo in robotics right now, today, is that most stereo algorithms can give you this coarse sense of what's around you and where you're going. And you can use it for downstream computations. In the world of stereo and research there are a lot of other issues that people are trying to deal with. How do you increase the density, increase the precision, deal with photometric issues like shiny objects, and deal with these differences between the images, just for the brightness, lack of texture. How do I deal with the fact that in some places they don't have lack of texture? How do I infer some sort of depth there? And also geometric ambiguities. It's like occlusion boundaries. How do I deal with the fact that occasionally there will be parts of the image that the left camera sees and the right camera doesn't see?

So there's ongoing research there. Most of these methods, I'll just say, try to solve stereo not as this local region-matching problem I mentioned, but as a global optimization problem. And so there's a lot of work in different global optimization algorithms. And there is hope that ultimately stereo will get to the point that it really can do the sort of thing I mentioned. I'm gonna pick this up and I really want to get a 3-D model and use that for manipulation. Probably not there today. I should just say that there's one simple way to get a huge performance boost out of your stereo algorithm, something that people often do. If anybody can guess one way to just take a stereo algorithm and make it work a whole lot better.

Think about how I could change this sheet of paper to be something that I could actually do matching on.

Student: Add texture.

Guest Instructor (Gregory Hager): You want to add texture. How could you add texture?

Student: Project lights.

Guest Instructor (Gregory Hager): There you go. Just put a little light projector on the top of your stereo system and you'll be amazed at how well it works. Suddenly this thing, which you couldn't match before, becomes the world's best place to do stereo because you get to choose the texture and you get to match it. So that's the other thing that people have looked a lot into. You'll see in the literature structure-like stereos. Another way to get better performance out of stereo. Okay. So that's stereo. Again, the message here is by using two cameras you can get at this point data density inaccuracy that still exceeds pretty much anything else you can imagine in the laser range finding world. It's not as reliable as laser range finding and that's probably the thing that is still the main topic of research. Any quick questions on that before I shift gears? Okay.

So I'm a robot, I'm running around, I've got real time stereo. I don't run into things anymore. If somebody walks in front of me I scurry away as quickly as I can, so that I don't hurt them. But I have no clue where I am in the world or I have no clue what's in the world around me. So if you said go over to the printer and get my printout, you know, where is the printer? Where am I? Where is your office? Who are you? So how can we solve those problems? And, in fact, this, I think, is an area of computer vision that I would say in the last decade has undergone a true revolution. Ten years ago if I would have talked about object recognition in this lecture we really had no clue. There are kind of some interesting things going on in the field, but we had no clue. And today there are people who would claim that at least certain classes of object recognition problems are solved problems.

We actually know very well how to build solutions and there are, actually, commercial solutions out there. So what is the problem with object recognition? Well, it's a chicken and egg problem. If I want to recognize an object there are many unknowns. So I look at an image. I don't know the identity of the object. I don't know where it is. And most importantly perhaps, I don't know what's being presented to me. So I don't happen to have a – oh, here we go. So if I say find my cell phone in the image you don't know if you're gonna see the front of the cell phone, the back of the cell phone, the top of the cell phone. You don't know if you're gonna see half of the cell phone hidden behind something else. You don't know what the lighting on the cell phone is gonna be.

Huge unknowns in the appearance let alone than finding it in the image and segmenting it and then actually doing the identification. So there's a sense in which if I could segment the object, if I could say here's where the object is in the image then solving recognition and pose would be fairly easy. Or if I told you the pose of the object solving segmentation and recognition would be easy to do or if I tell you what the object is finding it and figuring out its pose is easy to do. Doing all of them at the same time is hard. So for a long time people tried to use geometry. So maybe the right thing to do is to have a 3-D model of my cell phone and to use my stereovision to recognize it.

Now, we just said stereo's not real reliable, right? Probably not good enough to recognize objects. So another set of people said, well, how about if we recognize it from appearance? So let's just take pictures. The way I'm gonna recognize my cell phone is I've just got 30 pictures of my cell phone and you find it in the image. So you can do that and, in fact, can see here this is somewhere about ten years ago. They're doing pretty well on a database of about 100 objects. But you notice some other things not in the database. Black background. The objects actually only have, in this case, one degree of freedom. It's rotation in the plane. So, yes, they've got 100 objects, but the number of images they can see is fairly small.

No occlusion, no real change in lighting. So this is interesting. In some sense it will generate a lot of excitement because it's probably the first system that could ever do 100 objects. But it did it in this very limited circumstance. And so the question was, well, how do we bridge that gap? How do we get from hundreds to thousands and do it in the real world? Really the answer – you can almost think of it as combining both the geometry and the appearance-based approach. So the observation is that views were a very strong constraint. Giving you all these different views and recognizing from views work for a hundred objects, but it's just hard to predict a complete view. It's hard to predict what part of the cell phone I'm gonna see. It's hard to get enough images of it to be representative. It still doesn't solve the problem of occlusion if I don't see the whole thing.

So views seem to be in the right direction, just not quite there. So Cordelia Schmidt did a thesis in 1997 with Roger Moore where she tried a slightly different approach. She said, well, what if instead of storing complete views we store – think of it as interesting aspects of an object? If you were gonna store a face what are the interesting aspects? Well, the things like the eyes and nose and the mouth. The cheeks are probably not that interesting. There's not much information there. But the eyes and the nose and the mouth they tell you a lot. Or my cell phone, you've got all sorts of little textures.

So what if we just stored, if you think of it this way, thumbnails of an object? So my cell phone is not a bunch of images. It's a few thousand thumbnails. And now suppose that I can make that feature detection process very repeatable. So if I show you my cell phone in lots of different ways you get the same features back every time. Now, suddenly things start to look interesting because the signature of an object is not the image, but it's these thumbnails. I don't have to have all of the thumbnails. If I get half the thumbnails maybe I'll be just fine. If the thumbnails don't care about rotation in the plane that's good. If they don't care about scale even better.

So it really – it starts to become a doable approach and, in fact, this is really what has revolutionized this area. And, in particular, there is a set of features called SIFT, scale-invariant feature transform, which I know you've learned about in computer vision if you've had it. Developed by David Lowe, which, really, it become pretty much the industry standard at this point and, in fact, you can download this from his website and build your own object recognition system if you want. Let me just talk a little bit about a few details of the approach. So I said features is where we want to go. Well, the two things that we need to get good features. One is we need good detection repeatability. I need a way of saying there are features on this wall at this orientation, features on this wall at this orientation.

I should find the same features. So detection has to be invariant to image transformations. And I have to represent these features somehow. And probably just using a little thumbnail is not the best way to go, right? Because a thumbnail – if I take my cell phone and I rotate a little bit out of plane, or I rotate even in plane, the image changes a lot and I'd like to not have to represent every possible appearance of my cell phone under every possible orientation. So we need to represent them in an accordant and variant way and we need lots of them. And when I say lots I don't mean ten. I mean a thousand. We want lots of them.

So SIFT features solve this problem and they do it in the following way. They do a set of filtering operations to find features that are invariant to – the detection is invariant to rotation and scale. It does a localization process to find a very precise location for it. It assigns an orientation to the feature, so now if I redetect it I can always assign the same orientation to that feature and cancel for rotations in the image. Builds a key point descriptor out of this information and a typical image yields about 2,000 staple features. And there's some evidence that suggests to recognize an object like my cell phone you only need three to six of them. So from a few thousand features you only need 1 or 2 percent and you're there.

So, again, just briefly the steps. Set up filtering operations. What they're trying to do is to find features that have both a local structure that's a maximum of an objective function and a size or a scale that's a maximum of an objective function. So they're really doing a 3-dimensional search for a maximum. Once they have that they say, ah ha, this area's a feature. Key point descriptors. What they do is they can compute the so-called gradient structure of the image. So this allows them to assign a direction to features and so, again, by getting – by having an assigned orientation they can get rid of rotation in the image. So if you do that and you're running down an image you get confusing figures like this.

What they've done is they've drawn a little arrow in this image for every detective feature. So you can think of a long arrow as being a big feature, so a large-scale feature, and small arrows being fine detailed features. And the direction of the arrow is this orientation assignment that they give. And so you can see in this house picture it's not even that high of a res picture. It's 233 by 189 and they've got 832 original key points filtered down to 536 when they did a little bit of throwing out what they thought were bad features. So lots of features and lots of information is being memorized, but discreetly now. Not the whole image, just discreetly.

If you take those features and you try to match the features it turns out also they're very discriminative. So if I look at the difference in match value between two features that do match and two features that don't match it's about a factor of two typically. So there's enough signal there you can actually get matches pretty reliably. Now, I mentioned geometry. So right now an object would just be a suitcase of features. So if I'm gonna memorize my cell phone and I just said I gave you some thumbnails. So most systems actually build that into a view. So I don't just say my cell phone is just a bag of features, but it's a bag of features with some special relationship among them. And so now if I match a feature up here it tells me something about what to expect about features down there.

Or more generally if I see a bunch of feature matches I can now try to compute an object pose that's consistent with all of them. And so, in fact, that's how the feature matching works. Uses something called the hub transform, which you can think of as a voting technique. Just generally voting is a good thing. I'll just say as an aside, this whole thing that I've been talking about, all we're trying to do is set up voting. And we're really trying to be an election where we're not going down to the August convention to make a decision of who's winning the primaries. This is an election that we want to win on the first try. So these features are very good features. They do very discriminative matching of objects. You add a little bit of geometry and suddenly from a few feature matches you're getting rows and identity of an object from a very small amount of information.

So here are a couple of results from David Lowe's original paper. So you can see there's just a couple objects here. A little toy train and a froggy. And there is the scene. And if you're just given that center image I think even a person has to look a little bit before you find the froggy in the toy train. And on the right are the detected froggy and toy train. Including a froggy that's been almost completely hidden behind that dark black object. You just see his front flipper and his back flipper, but you don't see anything else. And the system's actually detected. In this case, two instances I think it – well, no, actually it's one. One box. So we've got one instance. So you've even got to realize that even though it's occluded it's one object out there.

So, I mean, these, again, I think it's fair to say a fairly remarkable result considering where the field was at that time. Since then there's been a cottage industry of how can we make this better, faster, higher, stronger? So this happens to be worth by Pons and Rothganger where they try to extend it by using better geometric models and slightly richer features. So they have 51 objects and they can have any number of objects in a scene. And this is the sort of recognition policy that you're starting to see now. So we're not talking about getting it right 60 percent of the time or 70 percent of the time. They're getting 90 plus percent recognition rates on these objects.

Now, of course, I've been talking about object recognition. Just point out that you can think of object recognition as there's an object in front of you and I want to know its identity and it's pose or you can think of the object as the world and I want to know my pose inside this big object called the world. So, for example, if I'm outside I might see a few interesting landmarks and recognize or remember those landmarks using features. And now when I drive around the world I'll go and I'll look for those same features again and use them to decide where I am. And so, in fact, this is, again, worked out of UBC where they're literally using that same method to model the world, recognize – I'm in this room and I see a bunch of features. I go out in the hallway I see a bunch of features. I go in the AL lab I see a bunch of features.

Store all those features and now as I'm driving around in the world I look for things that I recognize. If I see it then I know where I am relative to where I was before. Moreover, remember I said that for stereo to get geometry we didn't have to actually calibrate our stereo system. We could have one camera and it could just walk around. We could compute this so-called epipolar geometry automatically and then we could do stereo. So, in fact, what they've done in this math is they've from one camera as they're driving around they're not computing just the identity, but the geometry of all these features around them. So they can build a real 3-D map, just as I could build with the laser range finder, but now just by matching features and images.

And, in fact, we edited a joint issue of IJCD and IJRR about, I guess, six, eight months ago, and probably half the papers in that special issue ended up being how can you use this technique to map the world in different variations and flavors? So, again, it's a technique, which really in many ways is there. You can download it from the web practically and put it on your mobile robot and make it run. And, in fact, this is my favorite result. So this is the work of Ryan Eustis who was a post doc at Hopkins with Wes Whitcomb that does a lot of underwater robotics. So this is the Titanic, this is actually a Bowat expedition where they flew over the Titanic with a camera and the goal was obviously to get a nice set of images of the Titanic.

But the problem is that underwater it's really difficult to do very precise localization and odometry. And so what Ryan did was he took these techniques and he built effectively a mapping system, very high precision mapping system, that was able to take these images, use the images to localize the underwater robot and then put the images together into a mosaic and so this is a mosaic of the Titanic as they flew over it. You can see the numbers up there. They actually ran for about 3.1 kilometers. They have – what is it? How many images? There it is. Over 3,000 images, 3,500 images matched successfully, computed the motion of the robot successfully, filtered all of this together and were able to produce this mosaic. So really an impressive, impressive system.

Student:[Inaudible]

Guest Instructor (Gregory Hager): They actually have – so from this you, actually, do get the 3-D geometry. In this mosaic they've basically projected it down, but they are computing up to this set of features that they're able to use the 3-D geometry. And, actually, the little red versus brown up there, the red, I believe, is the original odometry that they thought they had on the robot and the brown is actually the corrected odometry that they computed or vice versa. I don't know which is which now. I can't remember if they did the two separate pieces or they did just one piece, but really impressive work. Very nice. Also, it mentions here doing a comment filter that he built a special purpose comment filter that operated on the space of reconstructed images.

So that is kind of the next piece of this puzzle or this, I guess I had one other three in here. So a lot of people are interested in 3-D now. So this is Peter Allen also putting together images. Here he's just showing the range data, but you can imagine if you have range in appearance now you can actually do interesting things using both 3-D in appearance information. I know there's some work going on here at Stanford in that also. So that's kind of Chapter 2. So now a set of techniques that not only let me avoid running into things in the world, but a set of techniques that let me say, well, where am I? And where is some things that I'm interested in?

So you can now actually imagine phrasing the problem I want to pick up the cell phone and you could actually have a system that recognizes the cell phone and is able to say, hey, there's the thing that I want to pick up. So I'll just finish up with what I thought was the last piece of this puzzle. Namely how do I pick it up? I'm not gonna tell you exactly how to pick it up. There's a lot of interesting and hard problems in figuring out how to put my fingers on this object to actually pick it up, but at least let's talk a little bit about the hand-eye coordination it takes for me to actually reach over and grab this thing or even better if I do that – I don't want to drop this. If I do that, how do I actually catch it again? Which luckily I did, otherwise it would be a very sad lecture.

So I'm gonna talk about this in two pieces. So one piece is gonna be visual tracking. So we're now really moving to the domain where I want to think about moving objects in the world and having precise information about how they're moving, how they're changing. So visual tracking is an area that attacks that problem and I know you saw a video maybe a week ago of a humanoid robot that was playing badminton or ping-pong or volleyball – volleyball I think it was. And so I think Professor Kateeb had all ready explained that they're doing some simple visual tracking of this big colored thing coming at them and using that to do the feedback.

Well, those big colored things are nice. Unfortunately my cell phone is not day-glow orange so its hard to just use color as the only thing that you can deal with, but tracking has been a problem of interest for a long time. Tracking people, tracking faces, tracking expressions, all sorts of different tracking. So what I think is interesting is first to say, well, what do you mean by tracking to begin with? It's kind of cool to write a paper that says tracking of X, but no one's ever defined what tracking is. So I have a very simple definition of visual tracking, which simply says I'm gonna start out with a target, you know, my face is gonna be the canonical target here.

So at time zero for some reason you've decided that's the thing you want to track. And the game in town is to know something about where it is at time T. And the something about where it is is something is what you in principle get to pick. So there's gonna be a configuration space for this object. I could – the simplest thing is your big orange ball. It's just round, so it's got no orientation. It just has a position in the image. So it's configuration is just where the heck is the orange ball? But you can imagine my cell phone has an orientation, so presumably orientation might be part of the configuration or if I start to rotate out of plane you get those out of plane rotations.

In fact, if it's a rigid object how many degrees of freedom must it have? They better know the answer to this.

Student: Six.

Guest Instructor (Gregory Hager): Six, yes. Believe me there’s no trick questions and he knows what he’s doing, so if he told you it’s six it really is six. There’s no question there. Six. So, sure, if this is a rigid object in principle there must be six degrees of freedom that describe it. Though, of course, if it’s my arm then it’s got more degrees of freedom. I wonder how many more it has. Anyway, okay. So this is gonna be a configuration space for this object and ultimately that’s what we care about is that configuration space. The problem is that the image we get depends on this configuration space in some way. And so here I’m gonna imagine for the moment that I can predict an image if I knew its configuration, if I knew the original image.

So you can imagine this is like the forward kinematics of your robot. I give you a kinematic structure and I give you some joint values and now you can say ah ha, here’s a new kinematic configuration for my object. So the problem then is I’m gonna think of a tracking problem. So I know the initial configuration. I know the configuration at time T. I know the original image. Now, what I’d like to do is to compute the change in parameters or even better just the parameters themselves. I don’t know what D stands for, so don’t ask me what D stands for. I want to compute the new configuration at time T plus one from the image at time T plus one and everything else I’ve seen.

Or another way to think of this is look, I’ve said I believe I can predict the appearance of an object from zero to T. I can also think of it as the other way around. I can take the image at time T and if I knew the configuration I could predict what it would have looked like when we started and now I can try to find the configuration that best explains the starting image. So this really is effectively a stabilization problem. I’m gonna try to pick a set of parameters that always make what I’m predicting look as close to the original template as possible. So in this case I’m gonna take the face and unrotate it and try to make it look like the original face. So my stabilization point now is an image.

And so this gives rise to a very natural sort of notion of tracking where I actually use my little model that I described, my prediction model, to take the current image, apply the current parameters, produce what’s hopefully something that looks like the image of the optic to start with. So if I started with my cell phone like this and later on it looks like that I’m gonna take that image, I’m gonna resample it so hopefully it looks like that again. If it doesn’t look like that there’s gonna be some difference. I’m gonna take that difference, run it through something. Hopefully that something will tell me a change in parameters. I’ll accumulate this change in parameters and suddenly I’ve updated my configuration to be right.

Now, what’s interesting about this, perhaps, was – just skip over this for the moment. So we can – for a pointer object we can use a very simple configuration, which turns out to be a so-called affie model. So how do I solve that stabilization problem? Well, again, I said I’m gonna start out with this predictive model, which is kind of like your kinematics, and if I want to go from a kinematic description talking about positions in space to velocities in space what do I use? Jacobian, imagine that. Hey, we’ve got kinematics in the rigid body world. We’ve also got kinematics in image space; let’s take a Jacobian.

If we take a Jacobian we're now relating what? Changes in configuration space to changes in appearance. Just like the Jacobian robotics, 308s, changes in configuration space to changes in Cartesian position. So there you go. So I'm gonna take a Jacobian; it's gonna be a very big Jacobian. So the number of pixels in the image might be 10,000. So I'm gonna have 10,000 rows and however many configurations. So 10,000 by six. So it's a very big Jacobian, but it's Jacobian nonetheless. And we know how to take those.

Well, now I've got a way of relating change in parameter to change in the image. Suppose I measure an error in the image, which is kind of locally like a change in the image. So an error in the alignment. Well, suppose I've effectively inverted that Jacobian. Now, again, I have to use a pseudo inverse because I've got this big tall matrix instead of a square matrix. Well, so I take this incremental error that I've seen in my alignment, go backwards to the Jacobian, and lo and behold it gives me a change in the configuration. And so I close the loop by literally doing an inverse Jacobian. In fact, it's the same thing that you use to control your robot to a position in Cartesian space through the Jacobian. Really no difference.

So it really is a set point control problem. Again, I won't go into details. Right now this is a huge, big time varying Jacobian. It turns out that you can show – and this is work that we did in – the name slips my mind. Assume you also did work showing that you can make this essentially a time and variance system, which is just a way of implementing things very fast. What does a Jacobian look like? Well, the cool thing about images is you can look at Jacobians because they are images. So this is actually what the columns of my Jacobian look like. So this is the Jacobian, if you look at the image, for a change in X direction, motion in X. And it kind of makes sense. You see its basically getting all of the changes in the image along the rows. Y is getting a change along the columns. Rotation is kind of getting this velocity field in this direction, so on and so forth.

So that's what a Jacobian looks like if you've never saw a Jacobian before. It turns out that what I had showed you is for plainer objects. You can do this for 3-D. So my nose sticks out a lot. If I were to just kind of view my face as a photograph and I go like this it doesn't quite work right. So I can deal with 3-D by just adding some terms to this Jacobian and, in fact, you'll notice – what can I say? I've got a big nose and so that's what comes out in the Jacobian in my face is my nose. Tells you which direction my face is pointing.

Again, we can deal with illumination and, this is actually probably a little more interesting, I can also deal with occlusion while I'm tracking because if I start to track my cell phone and I go like this, well, lo and behold there's some pixels that don't fit the model. So what I do is I add a little so-called re-weighting loop that just detects the fact that some things are now out of sync, ignore that part of the image. So you put it all together and you get something that looks not like that. So just so you know what you're seeing, remember I said this is a stabilization problem?

So if I'm tracking something, right? I should be stabilizing the image. I should be subtracting all the changes out. So that little picture in the middle is gonna be my stabilized face. I'm gonna start by tracking my face and this is actually the big linear system I'm solving. My Jacobian, which actually includes motion and includes some illumination components too, which I didn't talk about. So I'm just showing you a Jacobian. You can kind of see a little frame flashing. This is a badly made video. It's back when I was young and uninitiated. And so now you can see I'm running the tracking. This is just using planer tracking. So as I tip my head back and forth and move around it's doing just fine.

Scale is just fine because I'm computing all the configuration parameters that have to do with distance. I'm not accounting for facial expressions, so I can still make goofy faces and they come through just fine. Now, I'm saying to an unseen partner turn on the lights, so I think some lights flash on and off. Yep, there we go. So it's just showing you can actually model those changes in illumination that we talked about in stereo too through some magic that only I know and I'm not telling anyone. At least no one in this room. In fact, it's not hard. It turns out that for an object like your face if you just take about a half a dozen pictures under different illuminations and use that as a linear basis for illumination that'll work just fine for illumination.

Notice here that I'm turning side to side and it clearly doesn't know anything about 3-D. So you can actually make my nose grow like Pinocchio by just doing the right thing. So I'll just let this run a little longer. Okay. So, now what I did was I put in the 3-D model and so the interesting thing now is you see my nose is stock still, so I actually know enough about the 3-D geometry of the face in 3-D configurations that I'm canceling all of the configuration – all the changes due to configuration out of the image. As a side effect I happen to know where I'm looking too. So if you look at the backside and telling you at any point in time what direction the face is looking.

And here I'm just kind of pushing it. Eventually as you start to get occlusions it starts to break down, obviously, because I haven't modeled the occlusions. And I wish I could fast-forward this, but it's before the date of fast forwarding. Here my face is falling apart. He wasn't supposed to be there.

Student:[Inaudible]

Guest Instructor (Gregory Hager): It happens. It's a faculty at Yale saying hey, what are you doing? And this is just showing the – what happens if you don't deal with occlusion in vision. You can see that I'm kind of knocking this thing out and it comes back and then eventually it goes kaboom. And now we're doing that occlusion detection so I'm saying hey, what things match and what things don't match? And there you go. Cup, sees a cup. Says that's not a face.

So you can take these ideas and you can then push them around in lots of different ways. This is actually using 3-D models. Here we're actually tracking groups of individuals and regrouping them dynamically as things go on. Here is probably the most extreme case. So this is actually tracking two Davinci tools during a surgery where we learned the appearance of the tools actually as we started. So there is 18 degrees of freedom in this system. So it’s actually tracking in an 18 degree of freedom configuration space during the surgery. Okay. Very last thing. I have ten minutes and I’m racing for home now.

So I can track stuff, cool. So what? It’s fun, but the thing I said I wanted to do eventually was to finally manipulate something. I want to use all this visual information and I want to pick up the stupid cell phone and call my friends and say the vision lecture is finally over in robotics. We can go out and do something else. But the question is, how do I want to do that? So I’ve got cameras. They’re producing all sorts of cool information. I’ve got a robot that I want to make drive around. Where do I drive it to or how do I drive it? So what should I put in that box? Any suggestions? You can assume I’ve got two cameras. So I’ve got stereo. I’ve got pretty much anything you’ve seen. What’s anybody think about – I don’t care which way you want to think of it. What could you put in that box? What information would you use and what would you put in the box?

It’s gonna be on the mid-term. The simplest thing you could imagine, right? Is I’ve got – if I said I’ve got two cameras I can actually, with those two cameras, measure a point in space. I can actually calibrate those cameras to the robot and so I could just say, hey, go to this point in space. End of story. Good thing? Bad thing? Good or bad thing? Anybody think of why it could be good or bad? Yeah?

Student: It’s bad because if you run into anything on the way then you can’t really accommodate for that.

Guest Instructor (Gregory Hager): Right. So you’re not monitoring. Monitoring real time, right? So that would get rid of at least that problem. What if my robot’s not a real stiff robot? What if my kinematics aren’t great? Like turns out the Davinci kinematics aren’t great. So I could reach out to a point in space, but maybe my arm goes here or there instead, right? So the cameras tell me to go somewhere, but it’s not really closing the loop. So what if I do one better? What if I compute the position of my finger, track it, let’s say, and I compute the position of the phone in 3-D space. Now I can actually close the loop. I can say I want to make this distance zero and we could write down a controller that would actually do that.

Pick your favorite controller. I know Asama has some ideas of what they should be, but pick your favorite. And that will work. And, in fact, that will work pretty darn well it turns out. But suppose that my cameras are miscalibrated. And, in fact, suppose that I say, well, what I want you to do is to go along the line defined by the edge of the cell phone. I want you to be here for some reason. Turns out you can show that if you do it in position space or reconstructed space and your cameras aren’t perfectly calibrated you can actually get errors. In fact, you can get arbitrary errors if you want to. It’s not real likely, but it can happen. So there’s one other possibility, which is I’m looking at this thing and I’m looking at this thing, what if I close the loop in the image space?

What if I just write my controller on the image measurements themselves? Well, it turns out if you do that and what this is called it’s called encoding, so if you can encode the task you want to do like touch this point of the cell phone to my finger and do it in the image space, not in reconstructed space, well, you’ve defined an error that doesn’t mention calibration, right? It just says make these two things co-incident in the images. If you can close that loop stably, think of Jacobian, again, for example, you can actually drive the system to a particular point and you’ve never said anything about calibration in your error function, which means that even if the cameras are miscallibrated you go there.

In fact, there’s pretty good evidence that’s what you do. You don’t sit there and try to figure out the kinematics of your arm and the position in space and then kind of close your eyes and say go there. You’re watching, right? And you’re actually using visual space and we know this because I can put funny glasses on your eyes and after a while you still get pretty good at getting your fingers together. So, again, I’m running out of time. I won’t go into great detail, but the interesting question is really when can you do this in coding? When can I write things down in the image domain? And the answer, again, depends a little bit on what you mean by cameras, but suffice it to say you can do a set of interesting tasks just by doing things in the image space and closing the loop in the image space.

And the interesting fact is that a lot of, sort of, tasks that you might imagine, like putting a screwdriver on a screw or putting a disc in a disc drive, you can write it all on the image space and you don’t need to calibrate the cameras or you don’t need well calibrated cameras. I’ll have to say so this is – why did I ever get into this? Because I was sitting in this stupid lab at Yale and I started to do this tracking and just for the heck of it I built this robot controller to do visions. So you see I’m tracking and I’m controlling here. And like the usual cynical young faculty member, I never expected this thing to work the first time. I hadn’t calibrated the cameras, I just guessed what the calibration was. I just threw the code together and turned this thing on and it worked. I mean, it worked within a half a millimeter.

It wasn’t like it just worked. It was right. And then I started thinking about it and I realized, of course, it worked. I didn’t need to calibrate the cameras and so then we actually spent the next few years figuring out why it was that I could get this kind of accuracy out of a system where I literally put the cameras down on the table, looked at it, and said I think they’re about a foot apart, and ran the system. And this is the moral equivalent of that. So it’s out there doing some stuff and I’m doing the moral equivalent of pulling your eye out of your head and moving it over here and saying, okay, see if you can still do whatever you were doing.

And just to prove you can do useful things with it we had to actually do something with a floppy, so there you go. You can also see how long ago this was by the form factor of the Macintosh then putting the floppy disc into. Anyway, all right. So I’m about out of time, but I hope what I’ve convinced you of is that at least a lot of these basic capabilities we’ve got. We’ve got stereo, real time, gross geometry, we can recognize objects, we can recognize places, we can build maps out of it, we can track things, and we even know how to close loops in a way that are robust, so we don’t have to worry about having finely tuned vision systems to make it work.

So why aren’t we running around with robots playing baseball with each other? Well, I’ve given you kind of the simple version of the world. Obviously if I give you complex geometry objects you haven’t seen before it’s not clear we really know how to pick up a random object, but we’re hopefully getting close to. A lot of the world is doing deformable. It’s not rigid. Lots of configuration space. How do I talk about tracking it or manipulating it? And a lot of things are somewhere in between. Rigid objects on a tray, which, yes, I could turn it like this, but it really doesn’t accomplish the purpose in mind.

So understanding those physical relationships. In the real world there’s a lot of complexity to the environment. It’s not my cell phone sitting on an uncluttered desk. Well, my desk would be – I’d be happy if it were that uncluttered. And I’m telling you to go and find something on it and manipulate it. So complexity is still a huge issue and it’s not just complexity in terms of what’s out there. It’s complexity in what’s going on. People walking back and forth and up and down. Things changing, things moving. So imagine trying to build a map when people were moving through the corridors all the time.

In fact, again, I know this is something of interest to your human computer interaction. I can track people, so now, in principle, I can reach out and touch people. What’s the safe way to do it? When do I do it? How do I do it? What am I trying to accomplish by doing so? How do you actually take these techniques, but add a layer, which is really social interaction, and say social interaction to the top of it. And I don’t know if you’ve noticed, but I think these are not just – there’s a research aspect to it, but there’s also a market aspect to it. At what point does it become interesting to do it? What’s the first killer ap for actually picking things up and moving it around?

It’s cool to do it, but can you actually make money at it? And then the last thing, and I – at the beginning I said this. The real question is when you’re gonna be able to build a system where you don’t preprogram everything. It’s one thing to program it to pick up my cell phone. It’s another to program it to pick up stuff and then at some point have it learn about cell phones and say go figure out how to pick up this cell phone and do it safely and by the way don’t scratch the front because it’s made out of glass.

So, again, there’s a lot of work going on, but I think this is really the place where I have to stop and say I have no idea how we’re gonna solve those problems. I know how to solve the problems I’ve talked about so far, but I think this is where really things are really open-ended. And there are a lot of cross cutting challenges of just building complex systems and putting them together so they work. So I’ll just close then by saying the interesting thing is all of this that I’ve talked about is getting more and more real. This chart, I’ll just tell you, is dating myself, but I built my first visual tracking system in my last year of grad school because I wanted to get out and I needed to get something done and it ran on something called a MicroVAX 2 and it ran at, I think, ten hertz on a machine that cost $20,000, so it cost me about $2,000 a cycle to get visual tracking to work.

So I still have that algorithm today, in fact. I just kept running it as I got new machines so those numbers are literally dollars per hertz, dollars per cycle of vision that I could get out of the system. So it went from $2,000 to when I finally got tired of doing it about seven years ago when it was down to 20 cents a hertz. So literally for pocket change I could have visual tracking up and running. So all of the vectors are pointed in the right way in terms of technology. Knowledge I think we've learned a lot in the last decade. I mean, it's cool to live now and see all of this stuff that's actually happening.

I think the real challenge is putting it together so if you look at an interesting set of objects and an interesting set of tasks like be my workshop assistant, which is something I proposed about seven years ago, that you could actually build something that would literally go out and say ah ha, I recognize that screwdriver and he said he wanted the big screwdriver, so I'll pick that up and I'll put it on the screw or I'll hand it to him or whatever and oh, I've never seen this thing before, but I can always figure out enough to pick it up and hand that over and say, what is this? And when he says it's a plier I'll know what it is. So I think the pieces are there is the interesting message, but nobody has put it together yet. So maybe one of you will be one of the people to do so.

So I think I'm out of time and I think I've covered everything I said I would cover, so if there are any questions I'll take questions including after class. Thank you very much.

Instructor (Oussama Khatib):Thank you so much.

[End of Audio]

Duration: 76 minutes