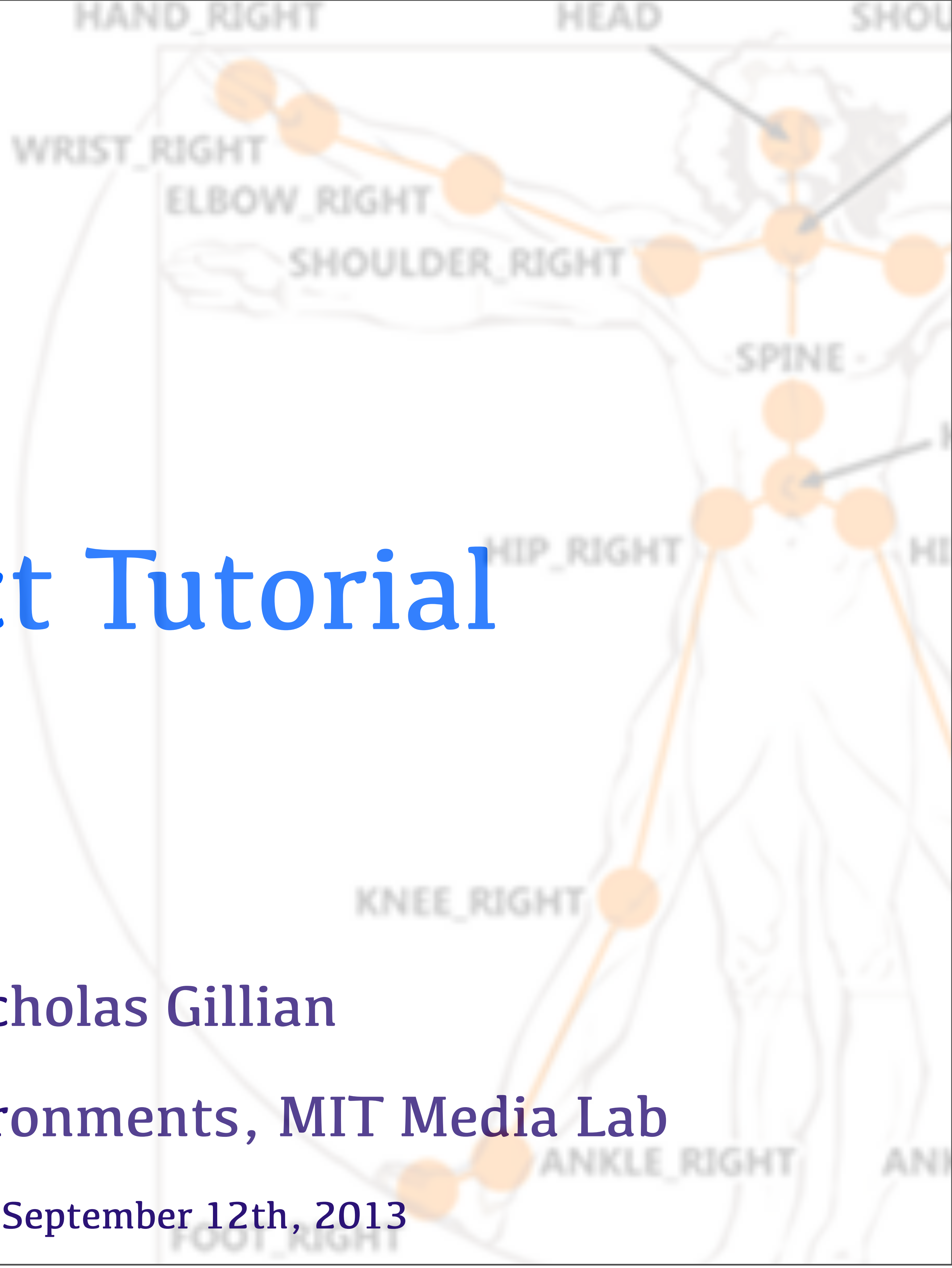


Kinect Tutorial

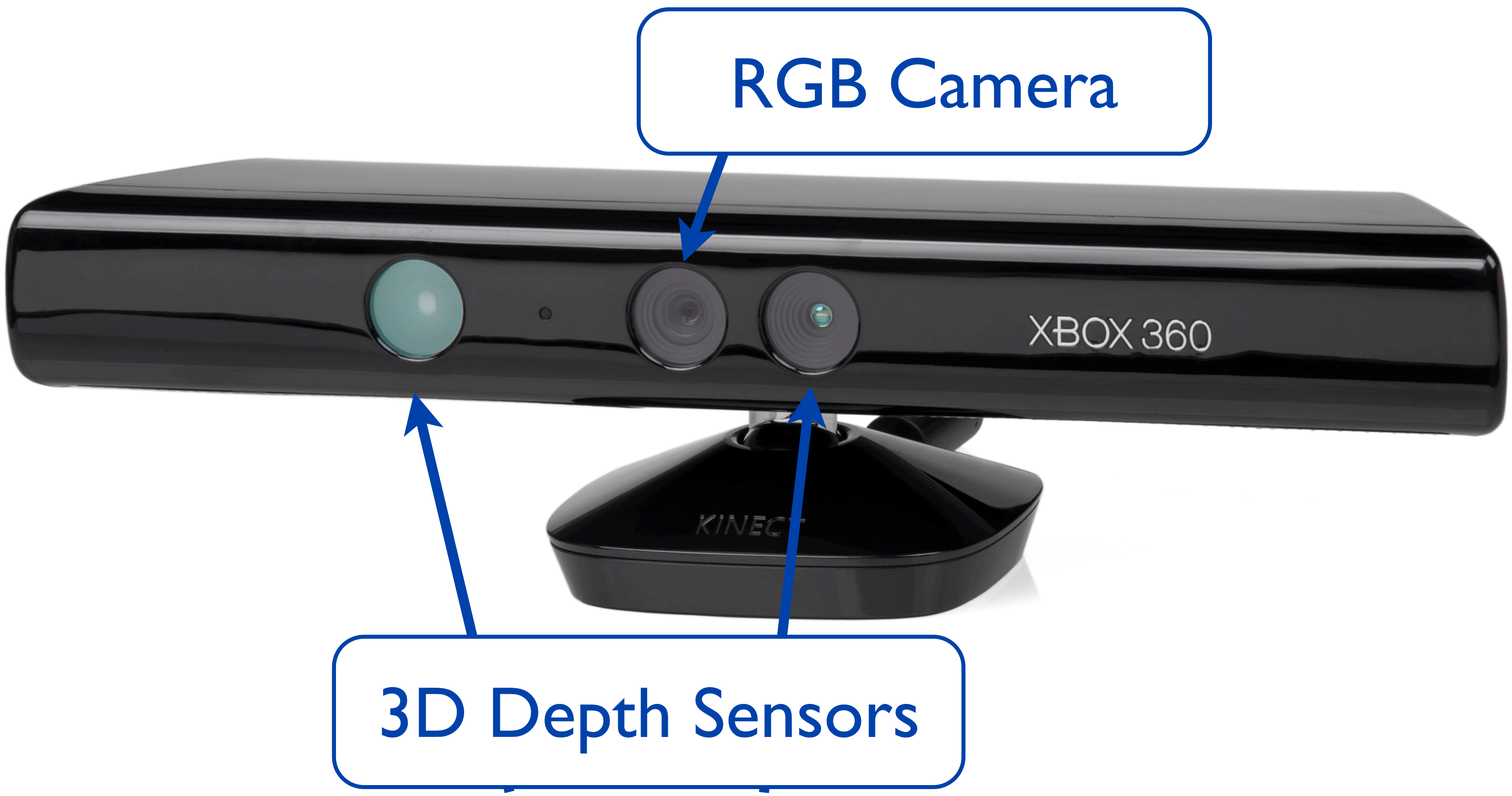
Nicholas Gillian

Responsive Environments, MIT Media Lab

Thursday, September 12th, 2013

Kinect

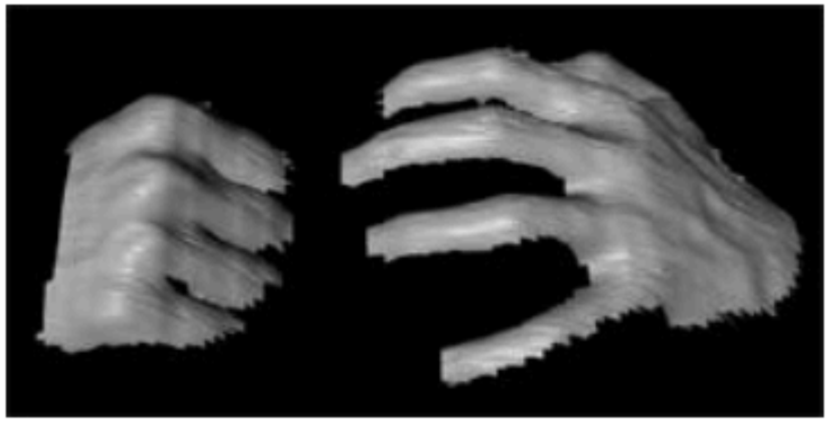
- Kinect uses structured light
- Body position is inferred using machine learning



- Depth map is constructed by analyzing a speckle pattern of infrared laser light

Kinect

- Structured light: project a known pattern onto the scene and infer depth from the deformation of that pattern



Zhang et al, 3DPVT (2002)

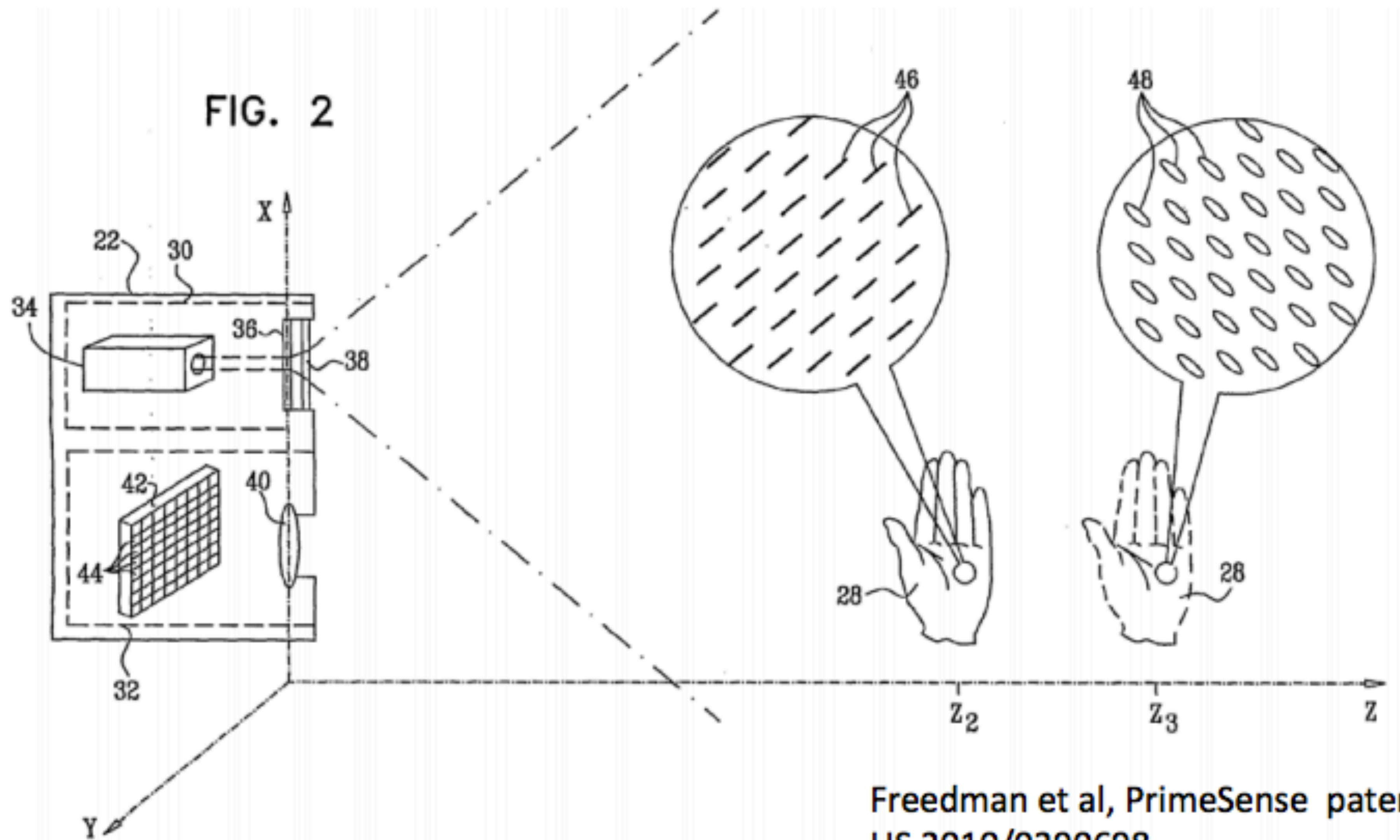
Jason Geng, Structured-light 3D surface imaging: a tutorial, Advances in Optics and Photonics, Vol. 3, Issue 2, pp. 128-160 (2011)

Kinect

- Kinect uses an astigmatic lens with different focal length in x- and y directions

- The lens causes a projected circle to become an ellipse whose orientation depends on depth

FIG. 2



Freedman et al, PrimeSense patent application
US 2010/0290698

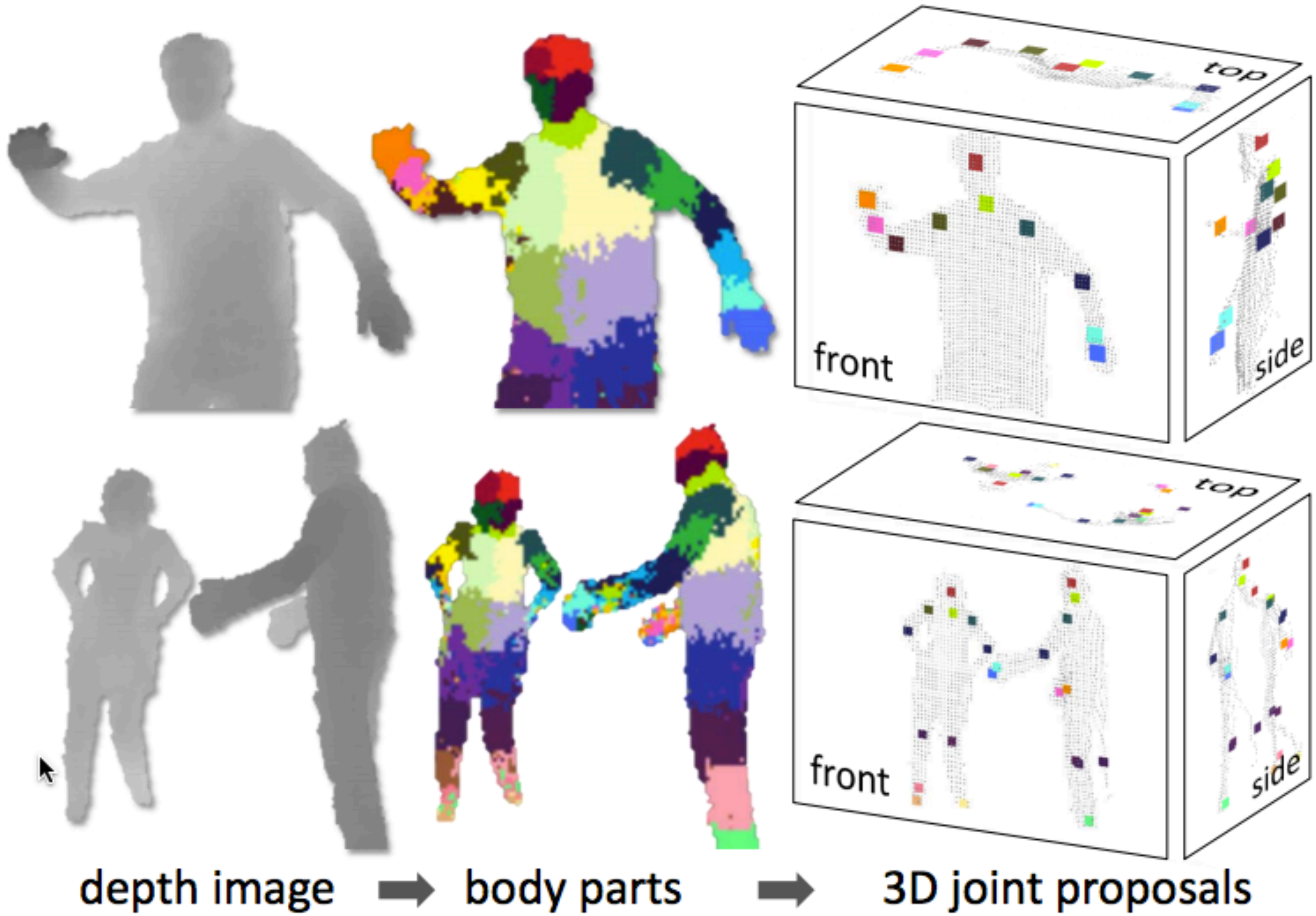
Kinect

- The Kinect also uses parallax, i.e. if you look at a scene from different angle, things that are closer get shifted to the side more than things that are far away

- The Kinect analyzes the shift of the speckle pattern by projecting from one location and observing from another

Kinect

- Body position is inferred using machine learning



Jamie Shutton et. al, Real-Time Human Pose Recognition in Parts from Single Depth Images, CVPR, 2011

Kinect

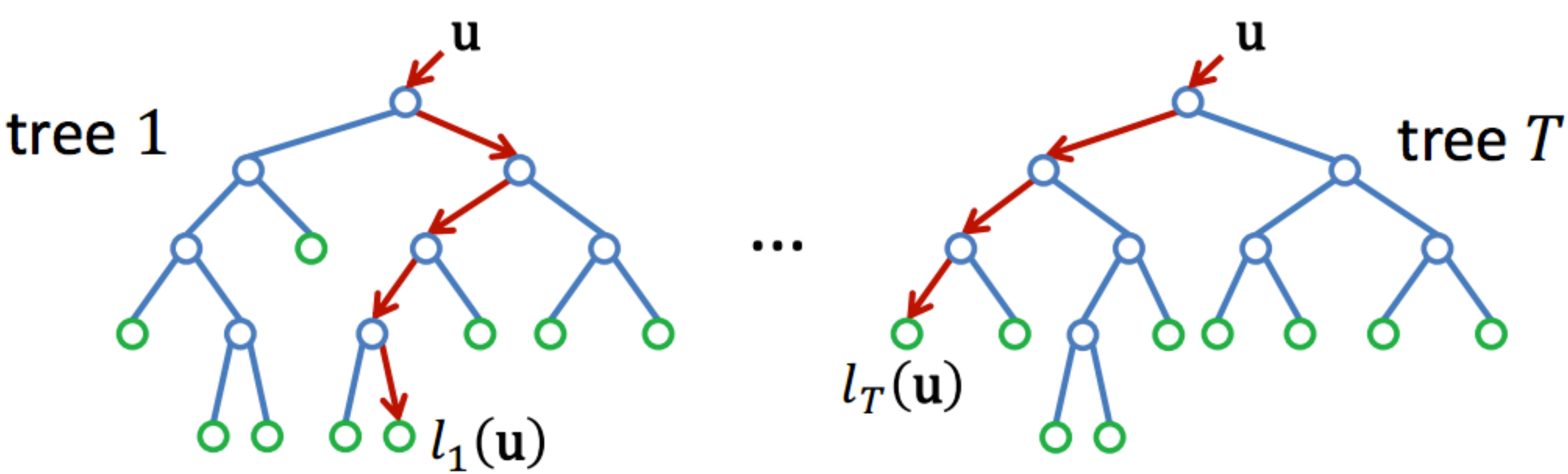
- Body position is inferred using machine learning

100K poses → 1 million training samples

Jamie Shutton et. al, Real-Time Human Pose Recognition in Parts from Single Depth Images, CVPR, 2011

Kinect

- Randomized Decision Forests



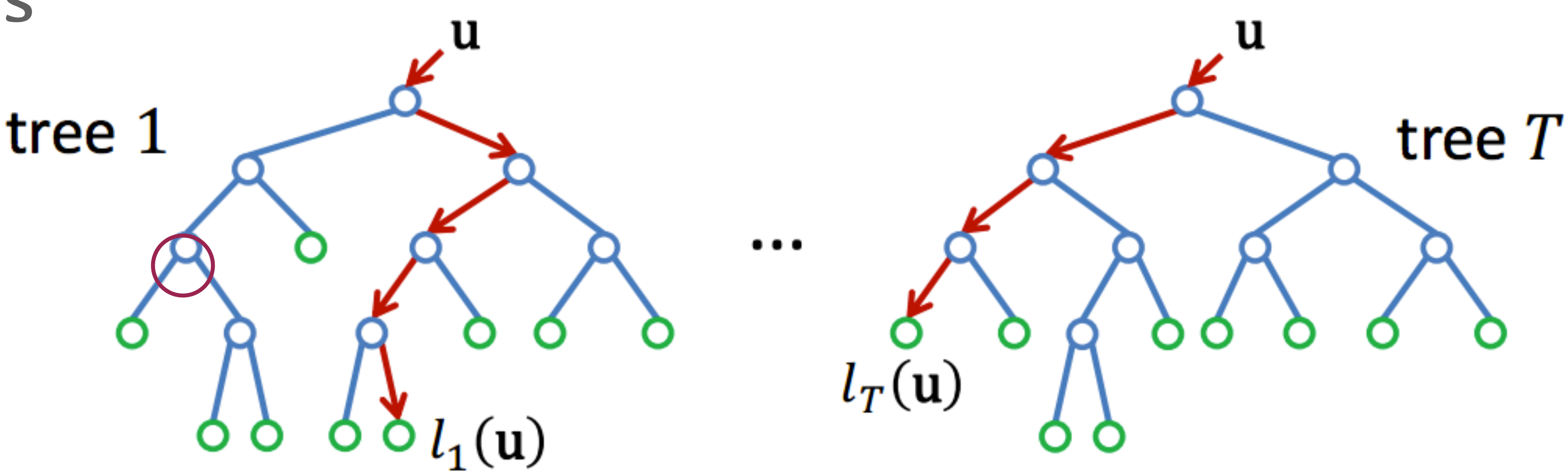
The probability of pixel u belonging to body part c is:

$$p(c|\mathbf{u}) = \frac{1}{T} \sum_{l \in \mathcal{L}(\mathbf{u})} p_l(c)$$

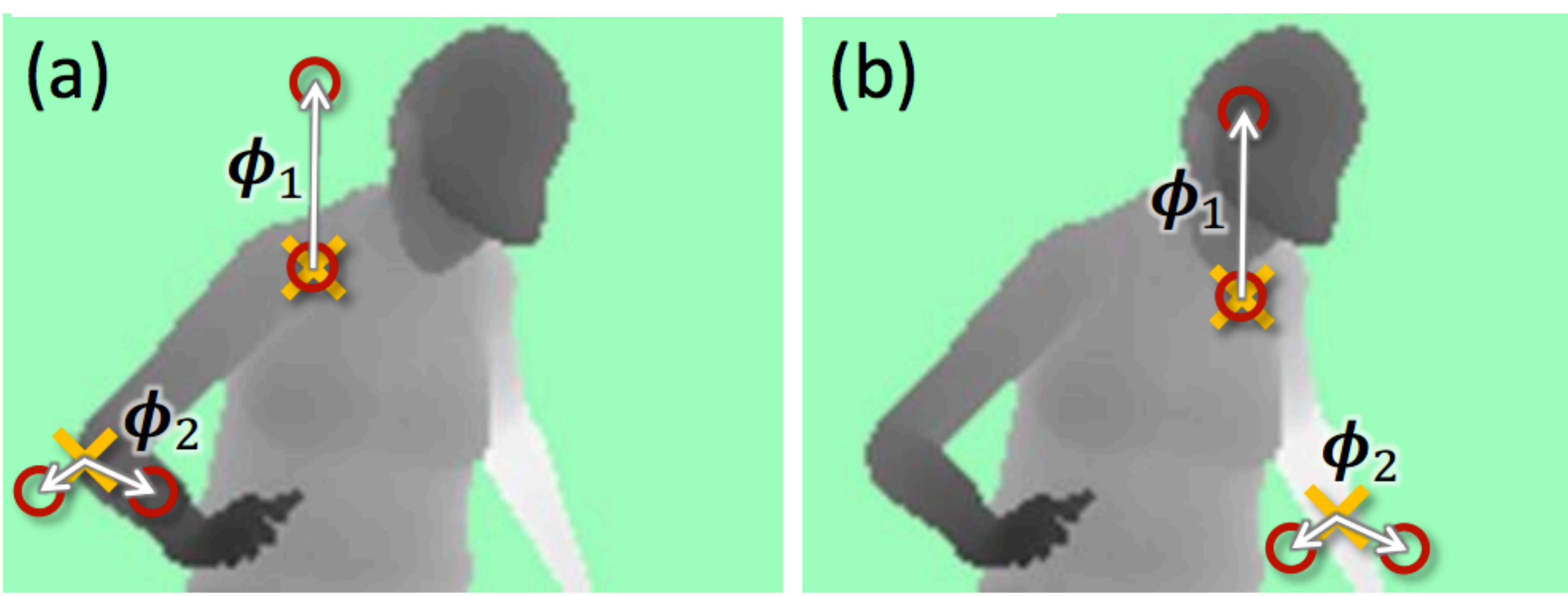
Jamie Shutton et. al, Real-Time Human Pose Recognition in Parts from Single Depth Images, CVPR, 2011

Kinect

- Features



$$f(\mathbf{u}|\phi) = z\left(\mathbf{u} + \frac{\boldsymbol{\delta}_1}{z(\mathbf{u})}\right) - z\left(\mathbf{u} + \frac{\boldsymbol{\delta}_2}{z(\mathbf{u})}\right)$$



Jamie Shutton et. al, Real-Time Human Pose Recognition in Parts from Single Depth Images, CVPR, 2011

Kinect Libraries, APIs & Tools

- Microsoft Official Kinect SDK
- OpenNI SDK
- Synapse
- Openframeworks

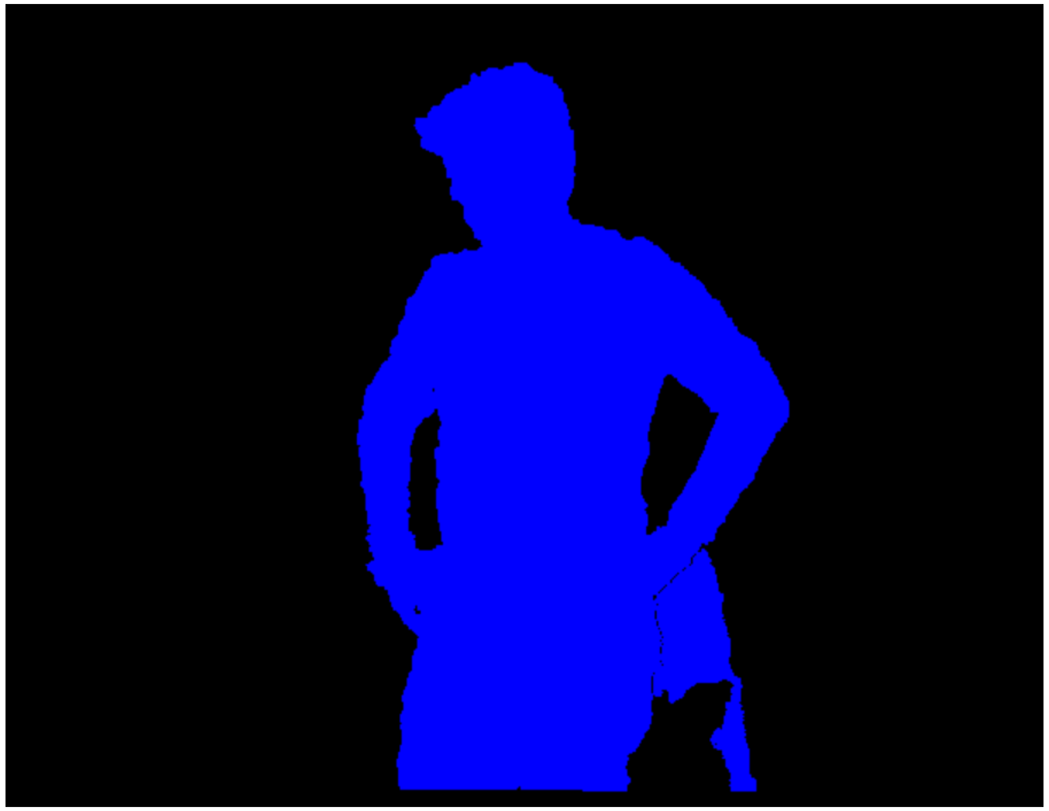
OpenNI - Installation

- OpenNI
- NITE
- Sensor Kinect

OpenNI - Data

RGB Image

Depth Image



Label Image

[640 480]

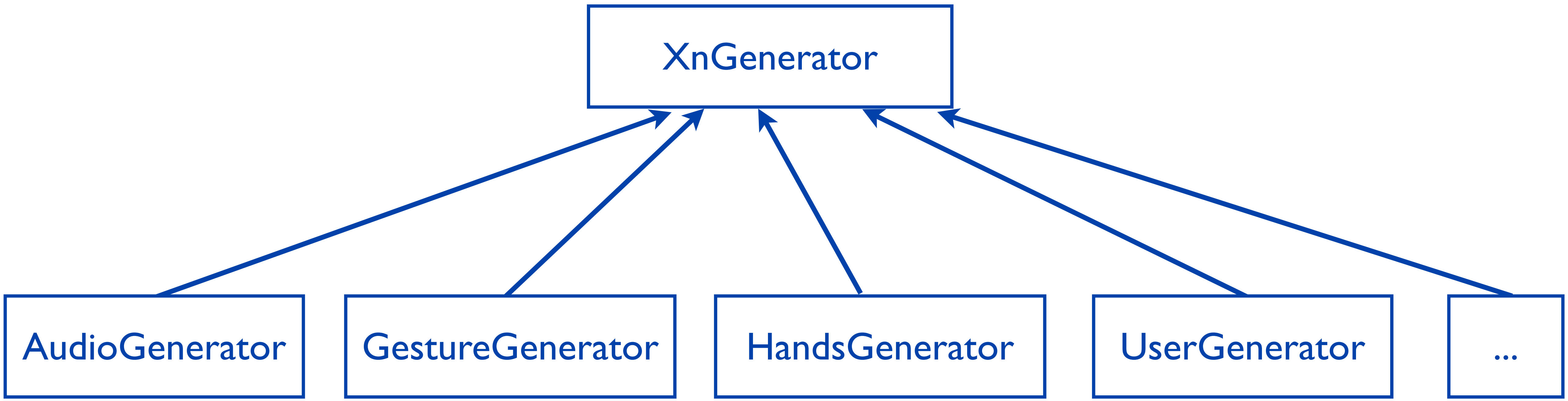
RGB: Unsigned char

Depth: Unsigned short

Label: Unsigned short

OpenNI™

OpenNI - Generators




```
//For example
DepthGenerator depthGenerator;
ImageGenerator imageGenerator;
```

OpenNI - Configuration

```
<OpenNI>
    <Licenses>
        <License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4="/>
    </Licenses>
    <Log writeToConsole="true" writeToFile="false">
        <!-- 0 - Verbose, 1 - Info, 2 - Warning, 3 - Error (default) -->
        <LogLevel value="3"/>
        <Masks>
            <Mask name="ALL" on="false"/>
        </Masks>
        <Dumps>
        </Dumps>
    </Log>
    <ProductionNodes>
        <Node type="Depth">
            <Configuration>
                <Mirror on="true"/>
            </Configuration>
        </Node>
        <Node type="Image" stopOnError="false" />
        <Node type="User" />
    </ProductionNodes>
</OpenNI>
```

This key is common PrimeSense key

Enable the depth camera

Enable the RGB camera

Enable the user tracker

OpenNI - Manual Configuration

```
XnStatus nRetVal = context.FindExistingNode(XN_NODE_TYPE_DEPTH, depthGenerator);
```

OpenNI - Manual Configuration

```
XnStatus nRetVal = context.FindExistingNode(XN_NODE_TYPE_DEPTH, depthGenerator);
if (nRetVal != XN_STATUS_OK){
        xn::MockDepthGenerator mockDepth;
        nRetVal = mockDepth.Create(context);

        // set some defaults
        XnMapOutputMode defaultMode;
        defaultMode.nXRes = 640;
        defaultMode.nYRes = 480;
        defaultMode.nFPS = 30;
        nRetVal = mockDepth.SetMapOutputMode(defaultMode);

        // set FOV
        XnFieldOfView fov;
        fov.fHFOV = 1.0225999419141749;
        fov.fVFOV = 0.79661567681716894;
        nRetVal = mockDepth.SetGeneralProperty(XN_PROP_FIELD_OF_VIEW, sizeof(fov), &fov);

        XnUInt32 nDataSize = defaultMode.nXRes * defaultMode.nYRes * sizeof(XnDepthPixel);
        XnDepthPixel* pData = (XnDepthPixel*)xnOSCallocAligned(nDataSize, 1, XN_DEFAULT_MEM_ALIGN);

        nRetVal = mockDepth.SetData(1, 0, nDataSize, pData);
        CHECK_RC(nRetVal, "set empty depth map");

        depthGenerator = mockDepth;
    }
```

OpenNI - Accessing Data

```
//Variables
UserGenerator userGenerator;
Context context;
```

```
// Read next available data
context.WaitOneUpdateAll( userGenerator );
userSelector->UpdateFrame();
```

OpenNI - Get the depth data

```
//Variables
DepthGenerator depthGenerator;
DepthMetaData depthMD;
```

```
//Get the latest depth and image meta data
depthGenerator.GetMetaData( depthMD );

//Get a pointer to the depth data
const XnDepthPixel* pDepth = depthMD.Data();

//Loop over the depth pixels
for (XnUInt y = 0;   y < depthMD.YRes();    y++){
        for (XnUInt x = 0;    x < depthMD.XRes();    x++){
                //Access the current depth value
                *pDepth;

                //Increase the depth pixel
                pDepth++;
        }
}
```

OpenNI - Get the RGB data

```
//Variables
ImageGenerator imageGenerator;
ImageMetaData imageMD;
```

```
//Get the latest image meta data
imageGenerator.GetMetaData( imageMD );

//Get a pointer to the image data
const XnRGB24Pixel* pImage = imageMD.RGB24Data();

//Loop over the image pixels
for (XnUInt y = 0;   y < imageMD.YRes();    y++){
        for (XnUInt x = 0;    x < imageMD.XRes();    x++){
                //Access the current pixel value
                * pImage;

                //Increase the pixel pointer
                pImage++;
        }
}
```

OpenNI - Skeleton data

```
//Variables
UserGenerator userGenerator;
```

```
XnUserID userIDs[ numTrackedUsers ];
XnUInt16 numUsers = userGenerator->GetNumberOfUsers();
userGenerator->GetUsers(userIDs, numUsers);

for( int i=0;    i < numUsers;    i++    ){
      if( userGenerator->GetSkeletonCap().IsTracking( userIDs[i] ) ){

            XnPoint3D userCenterOfMass;
            userGenerator->GetSkeletonCap().IsCalibrating( userIDs[i] );
            userGenerator->GetSkeletonCap().IsCalibrated( userIDs[i] );
            userGenerator->GetCoM( userIDs[i], userCenterOfMass );

            XnSkeletonJointTransformation jointData;
            if( userGenerator->GetSkeletonCap().IsJointAvailable( XN_SKEL_HEAD ) ){
                  userGenerator->GetSkeletonCap().GetSkeletonJoint(userIDs[i], XN_SKEL_HEAD,  jointData );
            }

      }
}
```

Kinect Gesture Recognition

OpenNI™

SUPERVISED LEARNING

Supervised Learning

Training Data

Supervised Learning

Training Data

$$\mathbf{X} = \{\{\mathbf{x}_1, \mathbf{t}_1\}, \{\mathbf{x}_2, \mathbf{t}_2\}, \{\mathbf{x}_3, \mathbf{t}_3\}, \ldots, \{\mathbf{x}_M, \mathbf{t}_M\}\}^T$$

Supervised Learning

Training Data

$$\mathbf{X} = \{\{\mathbf{x}_1, \mathbf{t}_1\}, \{\mathbf{x}_2, \mathbf{t}_2\}, \{\mathbf{x}_3, \mathbf{t}_3\}, \ldots, \{\mathbf{x}_M, \mathbf{t}_M\}\}^T$$

$$\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_N\}$$

Input Vector

Supervised Learning

Training Data

$$\mathbf{X} = \{\{\mathbf{x}_1, \mathbf{t}_1\}, \{\mathbf{x}_2, \mathbf{t}_2\}, \{\mathbf{x}_3, \mathbf{t}_3\}, \ldots, \{\mathbf{x}_M, \mathbf{t}_M\}\}^T$$

$$\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_N\}$$

Input Vector

{Feature Vector}

Supervised Learning

Training Data

$$\mathbf{X} = \{\{\mathbf{x}_1, \mathbf{t}_1\}, \{\mathbf{x}_2, \mathbf{t}_2\}, \{\mathbf{x}_3, \mathbf{t}_3\}, \ldots, \{\mathbf{x}_M, \mathbf{t}_M\}\}^T$$

$$\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_N\}$$

Input Vector

{Feature Vector}

Feature

Supervised Learning

Training Data

$$\mathbf{X} = \{\{\mathbf{x}_1, \mathbf{t}_1\}, \{\mathbf{x}_2, \mathbf{t}_2\}, \{\mathbf{x}_3, \mathbf{t}_3\}, \ldots, \{\mathbf{x}_M, \mathbf{t}_M\}\}^T$$

$$\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_N\}$$

Input Vector

{Feature Vector}

Feature

{Attribute}

Supervised Learning

Training Data

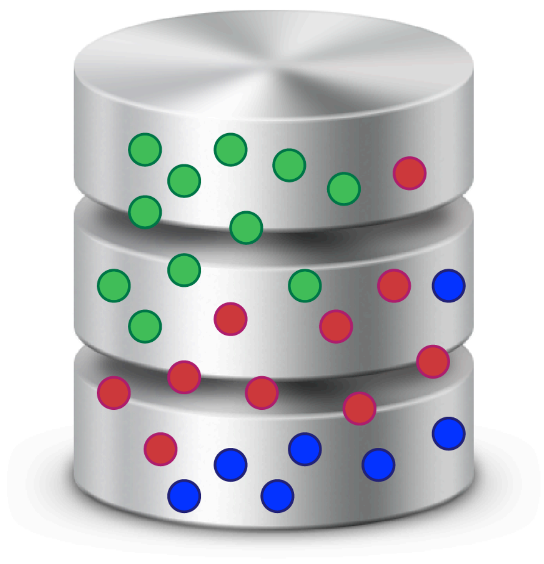
$$\mathbf{X} = \{\{\mathbf{x}_1, \mathbf{t}_1\}, \{\mathbf{x}_2, \mathbf{t}_2\}, \{\mathbf{x}_3, \mathbf{t}_3\}, \ldots, \{\mathbf{x}_M, \mathbf{t}_M\}\}^T$$

$$\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_N\} \qquad \mathbf{t} = \{k\}$$

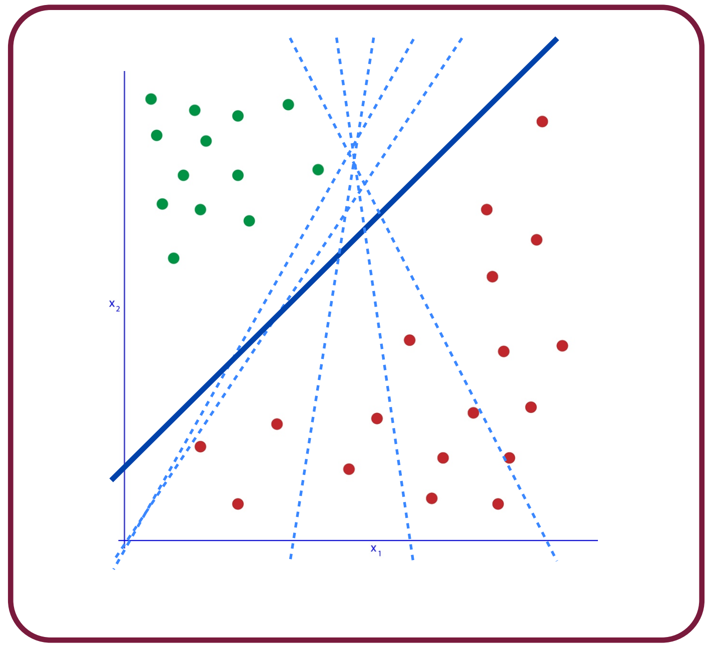
Input Vector

Target Vector

Supervised Learning



Training Data



Learning Algorithm

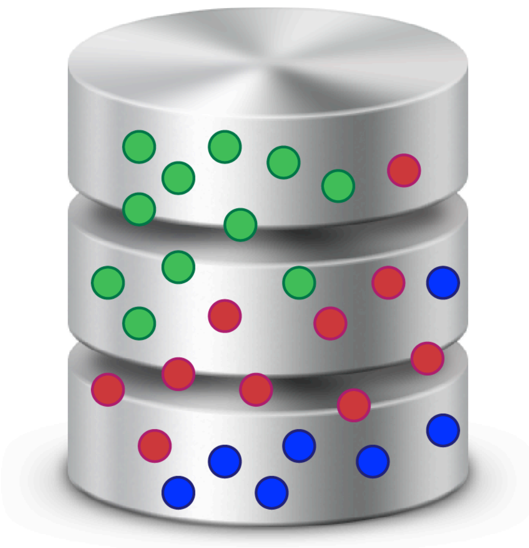
$$\mathbf{X} = \{\{\mathbf{x}_1, \mathbf{t}_1\}, \{\mathbf{x}_2, \mathbf{t}_2\}, \{\mathbf{x}_3, \mathbf{t}_3\}, \ldots, \{\mathbf{x}_M, \mathbf{t}_M\}\}^T$$

$$\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_N\} \qquad \mathbf{t} = \{k\}$$

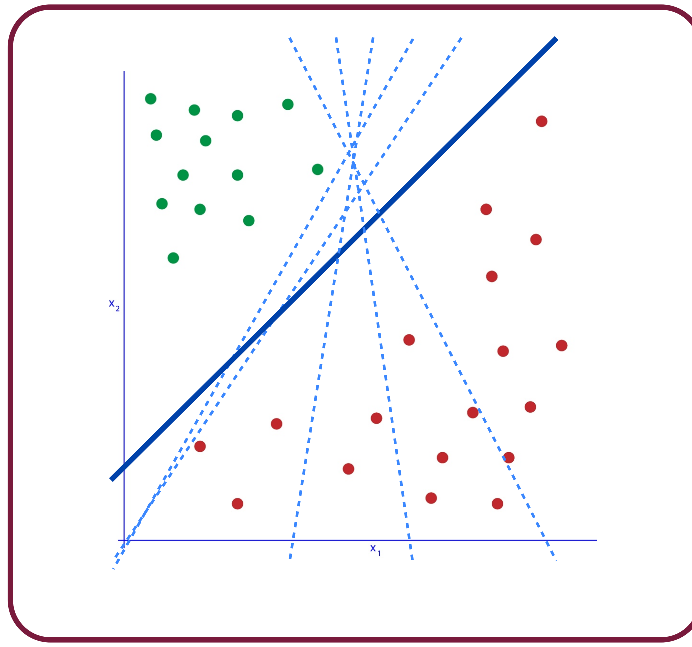
Input Vector

Target Vector

Supervised Learning

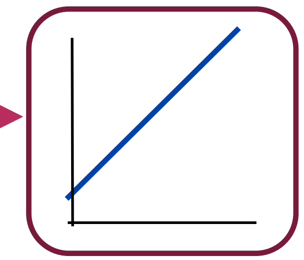


Training Data



Learning Algorithm

Model



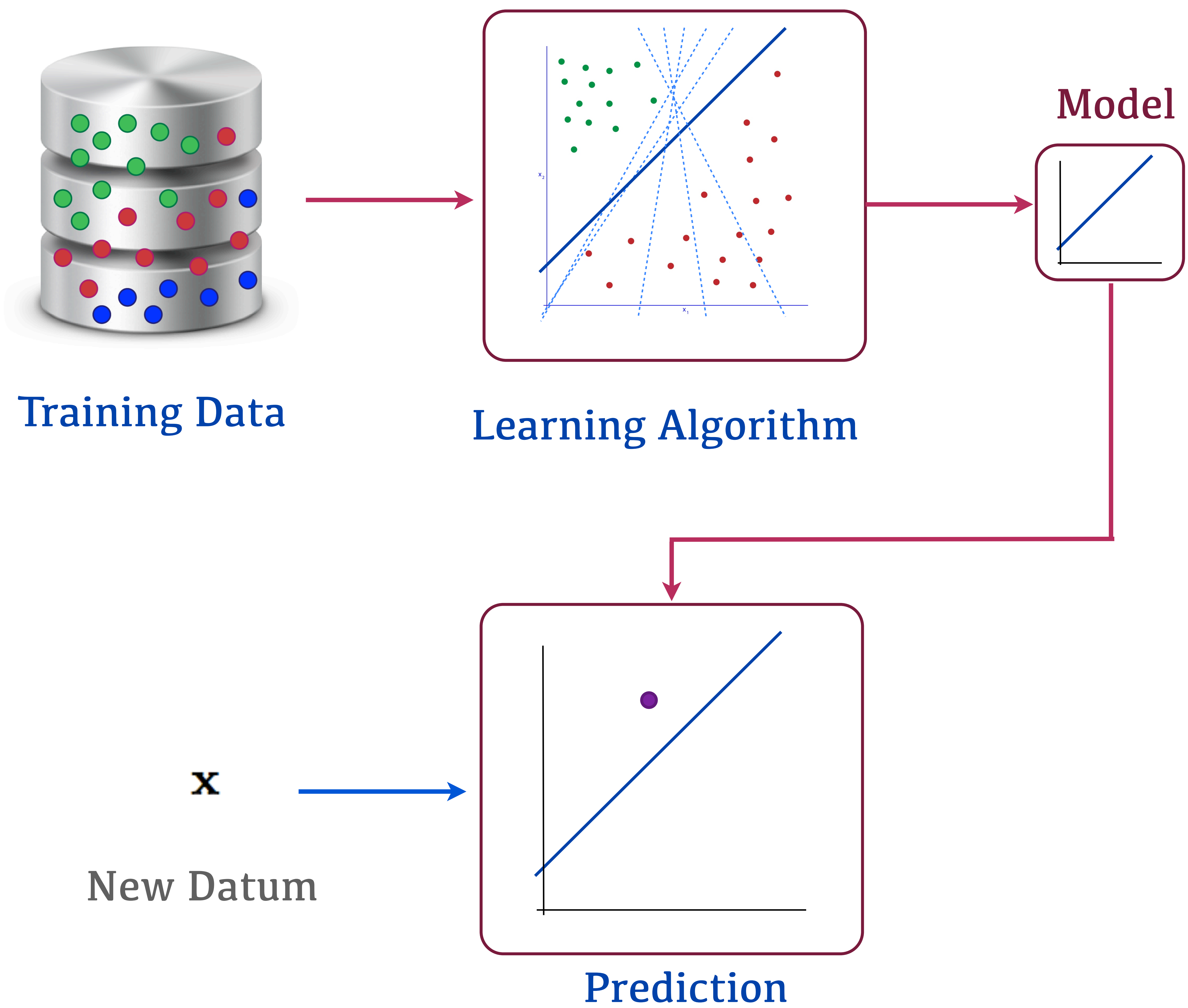
$$\mathbf{X} = \{\{\mathbf{x}_1, \mathbf{t}_1\}, \{\mathbf{x}_2, \mathbf{t}_2\}, \{\mathbf{x}_3, \mathbf{t}_3\}, \ldots, \{\mathbf{x}_M, \mathbf{t}_M\}\}^T$$

$$\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_N\} \qquad \mathbf{t} = \{k\}$$

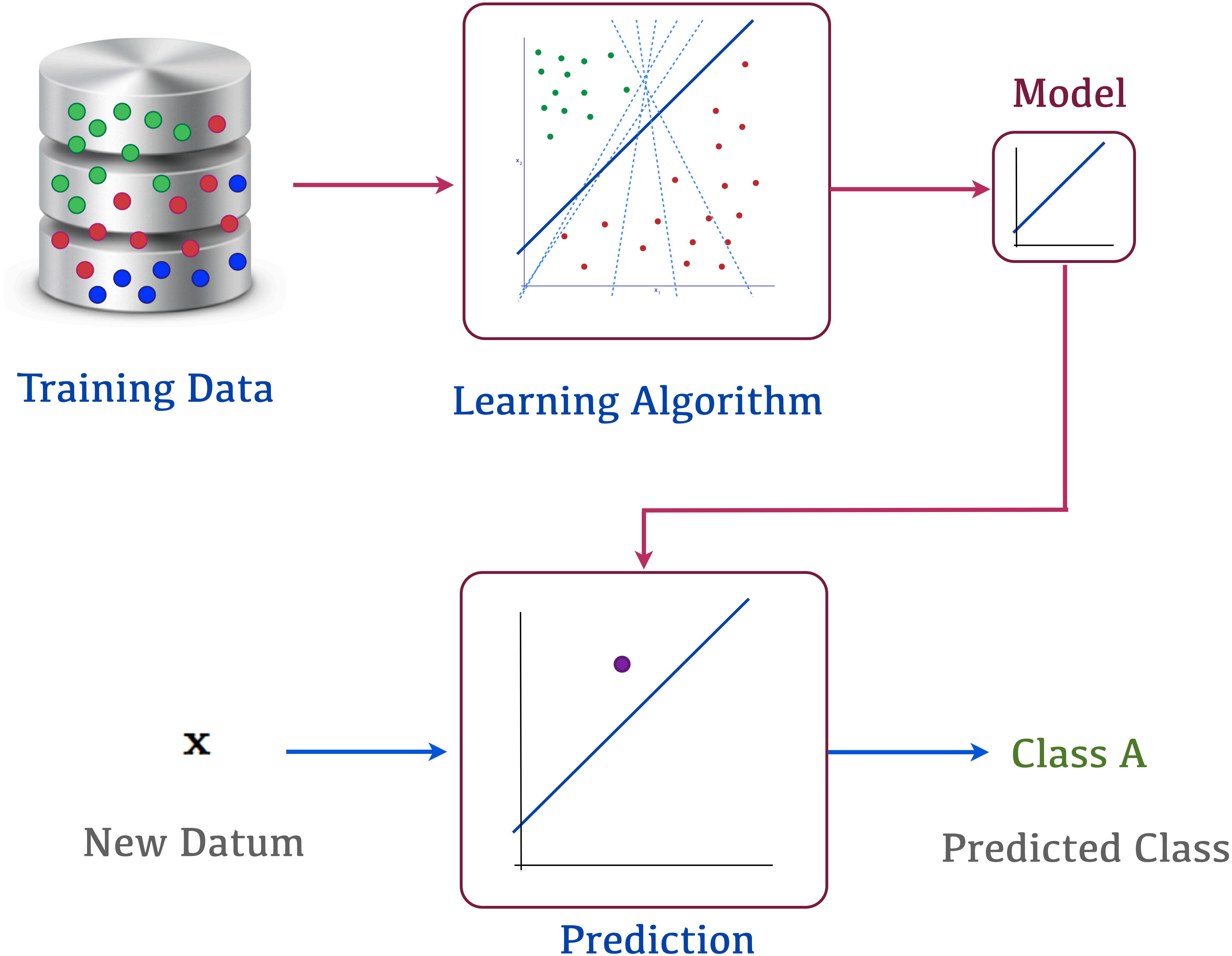
Input Vector

Target Vector

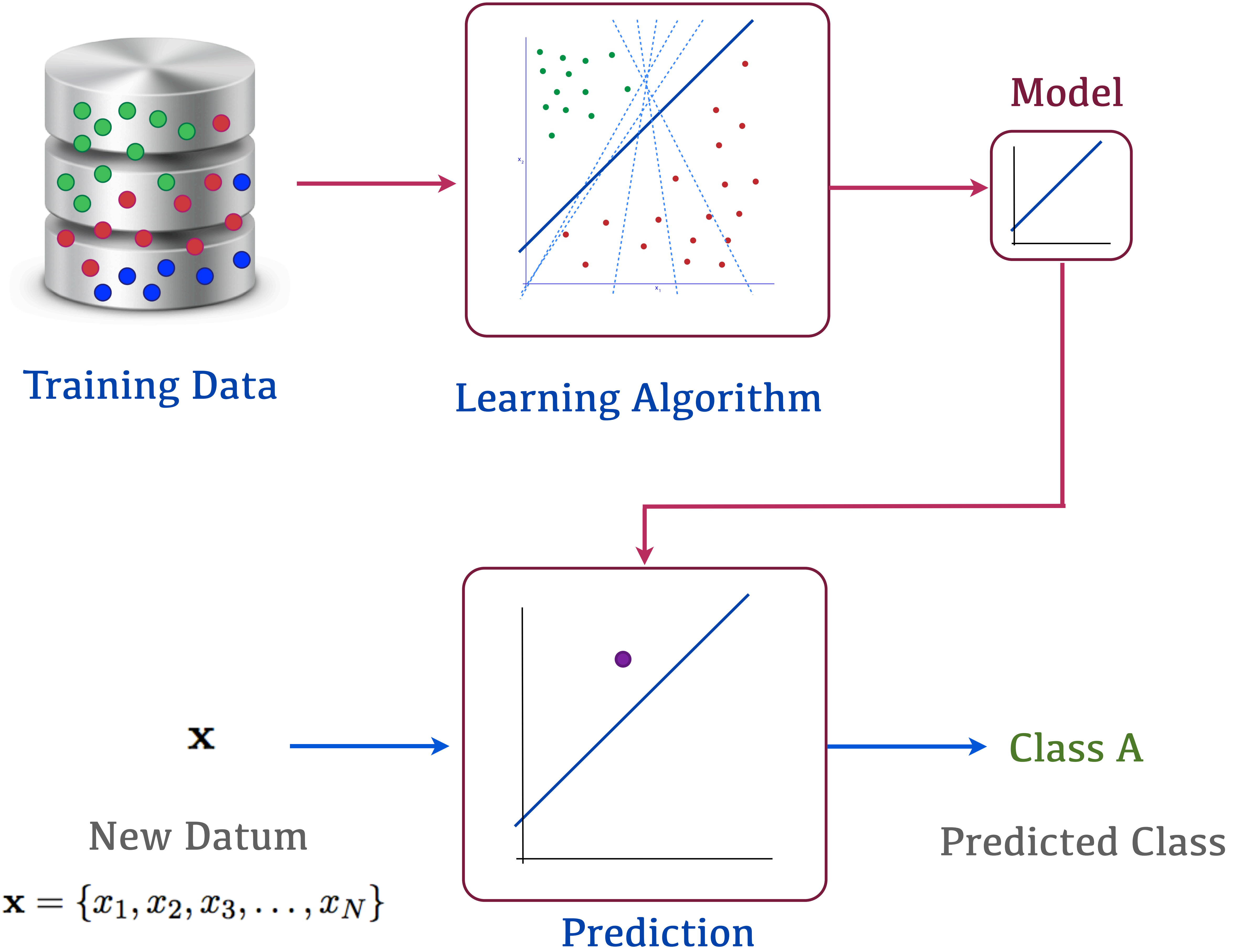
Supervised Learning



Supervised Learning

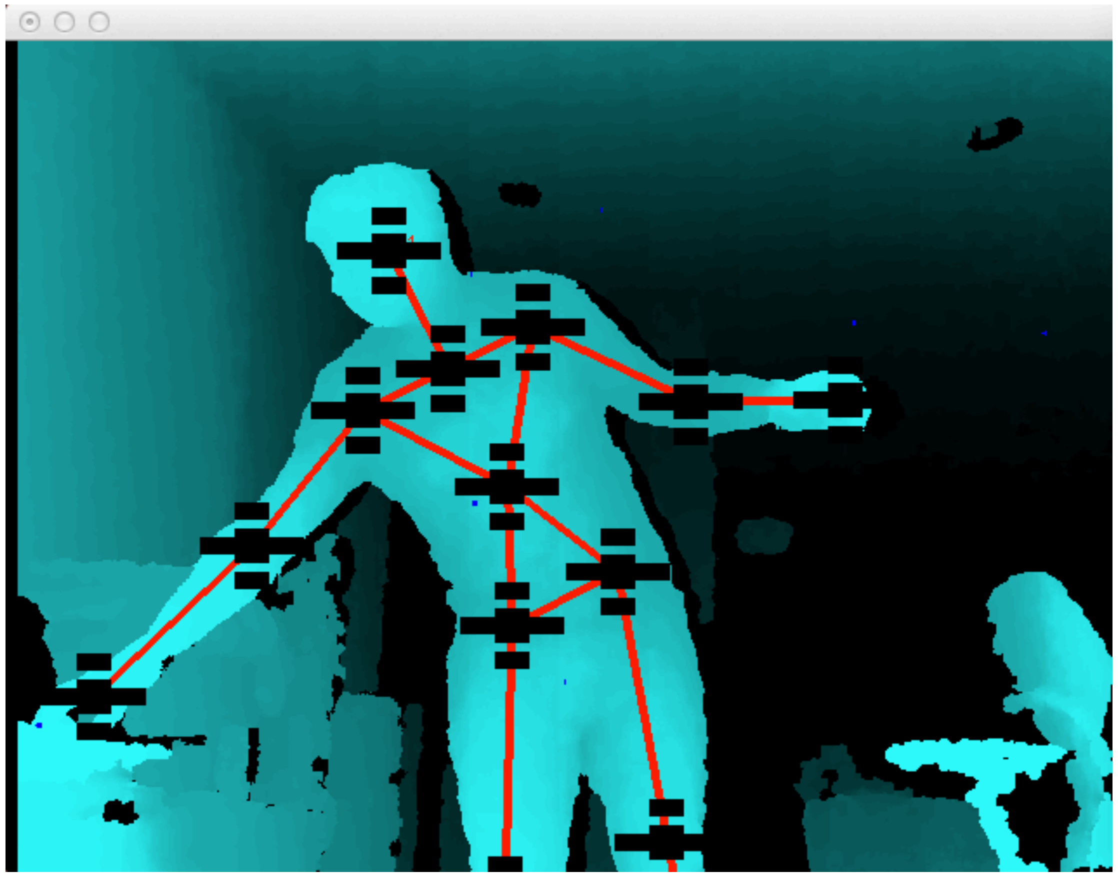


Supervised Learning



Kinect Gesture Recognition

Synapse

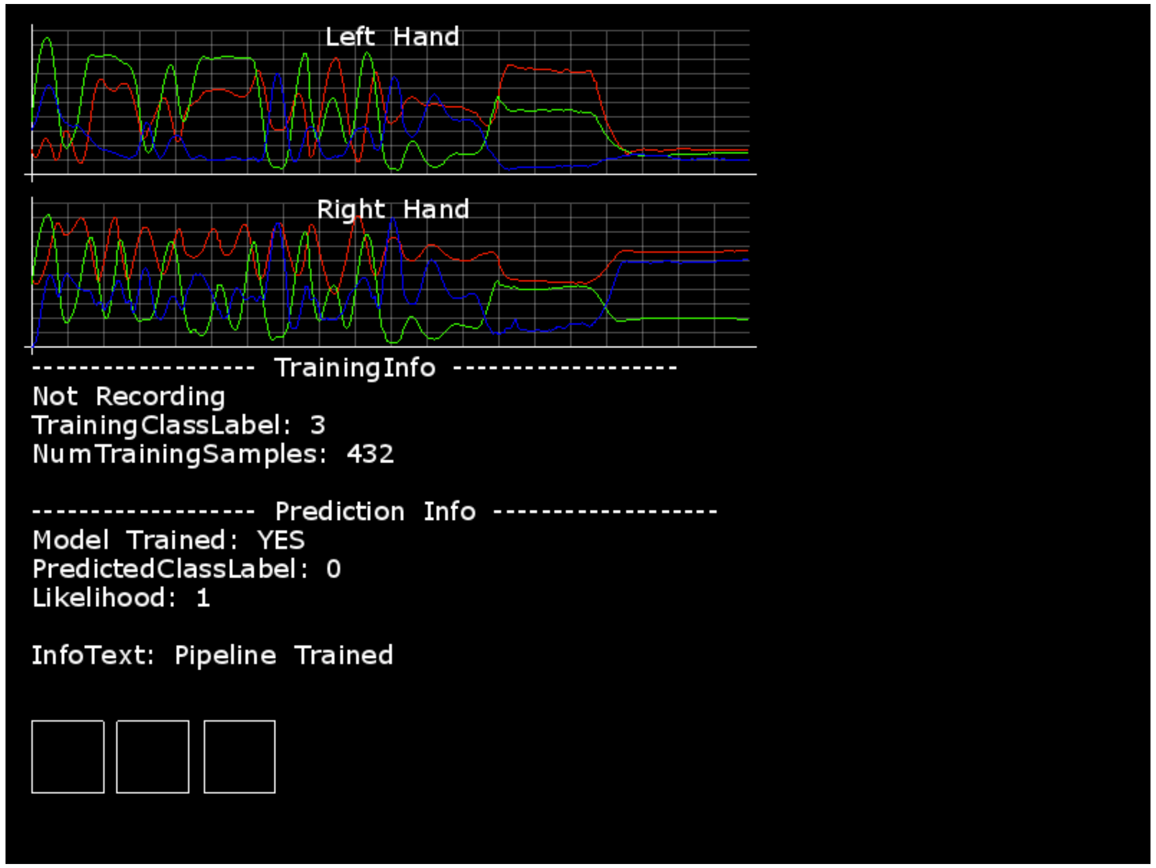


OpenNI

Skeleton Data

OSC

Openframeworks



GRT

www.nickgillian.com/wiki/pmwiki.php/GRT/OpenframeworksKinectExample

Kinect Code Tutorial Slides

www.nickgillian.com/09-12-13.html